Tekmicro

FastPMC/FastPCI Products

FastPMC/FastPCI - PIO
VxWorks Software Driver
User's Manual - Preliminary

TEK/TM-29480
March 2000

TEK Microsystems has made every effort to ensure that this manual is accurate and complete. However, TEK reserves the right to make changes and improvements to the products described in this manual at any time and without notice.

This product is covered by a limited warranty which is described in the manual. Other than the stated limited warranty, TEK disclaims all other warranties, including the warranties of merchantability and of fitness for a particular purpose. In the event of a failure of the hardware or software described in this manual, TEK's obligation is limited to repair or replacement of the defective item, or, if the item cannot be repaired or replaced, a refund of the purchase price for the item. TEK assumes no liability arising out of the application or use of the hardware or software, and assumes no responsibility for direct, indirect, incidental or consequential damages of any kind.

TEK Microsystems' products are not authorized for use as critical components in life support devices or systems without the express written agreement of an officer of TEK Microsystems.

Revision Information:

This manual describes version 1 of the tekpciPio VxWorks driver.

Document ordering code and release information:

URL: http://www.tekmicro.com/tm29480.pdf

TEK/TM-29480
March 2000

Table of Contents

Product Description

This software driver provides a VxWorks device driver API for TEK's PIO family of PMC, PCI and Compact PCI products.

All of TEK's PIO products use customizable FPGA hardware to implement the low-level interface to the PIO hardware. The software driver automatically downloads the appropriate FPGA image for the hardware model and interface being used. The user application is responsible for selecting the type of I/O interface which is desired(e.g. buffered I/O), which causes the software driver to select and download the appropriate FPGA image.

This software driver is intended to abstract the differences between the various types of PIO products.

This software driver API supports the following TEK Microsystems products:

- FPMC-PEI16-E, 16-bit Parallel ECL Input
- FPMC-PEI32-E, 32-bit Parallel ECL Input
- FPMC-PEI16-P, 16-bit Parallel PECL Input
- FPMC-PEI32-P, 32-bit Parallel PECL Input
- FPMC-PEO16-E, 16-bit Parallel ECL Output
- FPMC-PEO32-E, 32-bit Parallel ECL Output
- FPMC-PEO16-P, 16-bit Parallel PECL Output
- FPMC-PEO32-P, 32-bit Parallel PECL Output
- FPMC-PDIO32, Dual 16-bit EIA-485 Input/Output

This software driver API supports the following TEK protocols:

- Buffered I/O

Additional products and protocols are expected to be supported in future versions of this driver.

Support Information

License Information

This software driver and associated demonstration software are copyrighted program materials. The user is granted a license to use the program materials on a host computer only for the purpose of installing, configuring, operating and supporting TEK Microsystems' hardware products. The user is also granted a license to create derivative works based on the program materials for the purpose of integration of the user's host software with the program materials. Any other use, disclosure or copying of the program materials is expressly forbidden.

Warranty Information

This software driver and associated demonstration software are provided without a warranty of any kind. TEK expressly disclaims all product warranties, including the warranties of merchantability and of fitness for a particular purpose. TEK assumes no liability arising out of the application or use of the software programs, and assumes no responsibility for direct, indirect, incidental or consequential damages of any kind.

Contact Information

If technical support is required, please contact TEK through one of the following methods:

Internet	http://www.tekmicro.com
Email	support@tekmicro.com
Telephone	+1 781 270 0808
Facsimile	+1 781 270 0813
Mail	TEK Microsystems, Incorporated One North Avenue Burlington, MA 01803-3313

Revision History

For software driver revision information, please see the **CHANGES** file in the driver distribution.

Additional Documentation

The PIO modules use off-the-shelf components to implement several key functions, including the PCI bus interface, Serial EEPROM and customizable FPGA device. Although this driver attempts to abstract many of the interface details of these devices, review of the following additional documentation may be useful for proper operation and control of these products through this driver.

- PLX Technologies PCI 9080 Data Book
  Web site: http://www.plxtech.com
  URL: Requires registration with PLX Technologies
- Fairchild (formerly National) DM93CS46 Serial EEPROM
  Web site: http://www.fairchildsemi.com
  URL: http://www.fairchildsemi.com/ds/DM/DM93CS46.pdf
- Cypress ICD2053 PLL Clock Synthesizer
  Web site: http://www.cypress.com
  URL: http://www.cypress.com/pub/datasheets/icd2053b.pdf
- Altera Application Note AN116, Configuring FLEX 10K Devices
  Web site: http://www.altera.com
  URL: http://www.altera.com/literature/an/an116.pdf

The PDF versions of these data sheets may be accessed through the manufacturers' web pages as shown above. The URLs were valid as of the date of this manual.

Printed versions of the above data sheets are available from TEK's technical support department.

Installation

The software driver and demonstration programs are delivered as a TAR.GZ file. The driver distribution includes all required source and include files.

To install the file on a Solaris system, enter the following commands:

```
gunzip piodrv.current.tar.gz
tar xvf piodrv.current.tar
```

This will place all of the required files in the current project directory.

Running the command “make” in this directory will create the executable libraries along with a suite of demonstration programs.

Architecture

Device Driver Layers

The VxWorks software driver is implemented using a layered architecture:

- The lowest layer, contained in **tekpciVxWorks.c**, provides generic PCI services to the driver. This module is the only module that is Board Support Package (BSP) specific, although most of the files are CPU architecture specific due to the use of Big/Little Endian macros.
- The next layer, contained in the library file **tekpciPlx9080.obj**, provides interface routines to configure, control and monitor the PLX Technologies' PCI 9080 interface controller.
- The following layer, contained in **tekpciFlex.obj**, provides an intermediate layer to handle downloading of FPGAs and management of EEPROMS.
- The top layer, contained in the library file **tekpciPio.obj**, provides a set of interface routines, which configure, control and monitor products in TEK's PIO family.
- Additionally, **tekFpga** and **tekClockSyn** exist as generic utility modules for handling standard FPGA images and supporting clock synthesizers respectively.

The driver is supplied with a set of demonstration programs, which are described at the end of this document.

Data Structures

The primary data structure used by the device driver is the **TEKPCI_HANDLE** data structure, defined in **tekpci.h**. The **TEKPCI_HANDLE** structure is used as a "handle" to a hardware module. A user application may support any number of hardware modules by creating separate **TEKPCI_HANDLE** structures, one for each hardware module.

Text Messages

Some of the driver functions may be used to generate formatted text output, which is used to report driver, hardware or software status information. Driver functions, which generate text output follow a set of conventions which are designed to support different user operating environments.

Any function that generates screen output will take two arguments: *output* and *handle*. The *output* argument is a pointer to an output function that accepts strings and outputs them to the display. The *handle* argument is passed to the output function for each function call and may be used to manage different task, process or window contexts for the output.

This implementation isolates the driver's display functions from the actual screen output. The output function may be used capture the text and log it to a file, display it, or perform any other desired function.

If the *handle* argument is unused, it may simply be set to NULL. The meaning of the *handle* argument is determined by the user-supplied *output* function; the driver routines do not perform any operations with the *handle* argument except to pass it as an argument to the user-supplied *output* function.

For example, the **tekpciPlx9080IntDisplay** function has the following prototype:

```
STATUS tekpciPlx9080IntDisplay (
    TEKPCI_HANDLE pci,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

In a "generic" ANSI C environment, the following code may be used to implement the output function and execute the **tekpciPlx9080IntDisplay** function:

```
static void output_puts (void *handle, const char *s)
{
    printf ("%s", s);
    fflush (stdout);
}

STATUS display_int_status (TEKPCI_HANDLE pci)
{
    if (tekpciPlx9080IntDisplay (pci, output_puts, NULL) != OK) {
        return (ERROR);
    }

    return (OK);
}
```

None of the driver functions use the 'printf' function (or any other output functions) directly.

Note that the output function is required to be present (i.e. is not allowed to be NULL) unless explicitly stated. If no output is desired, the user may write an appropriate dummy function which simply discards the text output.

Text Message Control

Some of the driver functions accept an argument, sometimes referred to as the "verbosity" parameter, to control the amount of text to display to the user. Most of the Built-In-Test functions include this argument to allow the user application to tailor the amount of status information generated during a test. The "verbosity" parameter is provided using the **VERBOSE** enumerated type, defined in **tekStdTypes.h**. A **VERBOSE** type may be one of three states:

- **VERBOSE_NONE**. When this level is used, the driver function will generate no text output. The user may provide a NULL output function in this case.
- **VERBOSE_ERROR**. When this level is used, the driver function will generate text output only in the event of an error condition.
- **VERBOSE_ALL**. When this level is used, the driver function will generate test output as the function is executed.

Initialization

The device driver is expected to operate in an environment that does not perform PCI "Plug and Play" initialization. The following discussion assumes that the user application has a predefined hardware configuration and address map for the PCI hardware.

Before using any driver calls, each driver layer should be initialized by calling **tekpciInit( )**, **tekpciPlx9080Init( )** and **tekpciPioInit( )**.

Each instance of a **TEKPCI_HANDLE** is initialized as follows:

1. The user application allocates a **TEKPCI_HANDLE** data structure. The data structure may be statically or dynamically allocated. One **TEKPCI_HANDLE** structure is required for each hardware module controlled by the driver.

2. The user application clears the **TEKPCI_HANDLE** data structure. Typically, this is done using **memset( )**; for strict ANSI compliance, the pointer fields should be explicitly set to NULL.

3. The user application fills in the *bus*, *device*, *function* and (optionally) *slot* fields of the **TEKPCI_HANDLE** structure with the appropriate values for the PCI/PMC module. These values may be determined dynamically by scanning the PCI configuration space, or more typically, the user application will be designed to operate with a specific hardware module in a specific PCI/PMC slot.

4. The user application fills in the desired addresses for the *RegIoAddr* and *RegMemAddr* fields. These fields define the first two PCI base addresses in the host processor's address space; the first two PCI base addresses are used for I/O and memory space access to the PCI 9080 local control/status registers.

5. The user application calls the **tekpciMap( )** function to map the PCI device into the host processor's address space. The *RegIoAddrCard* and *RegMemAddrCard* arguments to **tekpciMap( )** should be the PCI bus addresses which correspond to the *RegIoAddr* and *RegMemAddr* pointers in the host processor's address space. The *MemAddrCard1* and *MemAddrCard2* arguments should be NULL at this stage.

6. Once the PCI 9080 controller is accessible to the host processor, the **tekpciPlx9080** driver functions may be used to access the Serial EEPROM and determine the specific hardware characteristics of the module.

7. The user application can then fill in the desired addresses for the *MemAddr1* and *MemAddr2* fields and call **tekpciMap( )** a second time. The *MemSize1* and *MemSize2* fields should be set to the maximum memory space to allocate.

PIO Driver API

The **tekpciPio** software driver provides functions which initialize PIO modules, read and write EEPROM information, download FPGA images, control interrupts, perform DMA transfers, control and monitor the data link hardware, and perform confidence testing of the module.

The **tekpciPio** API is intended to abstract the differences between different types of physical interfaces which implement streaming parallel I/O. In particular, the **tekpciPio** API defines a single set of streaming I/O functions which may be used for parallel ECL, parallel PECL and parallel EIA-485 hardware implementations.

The **tekpciPio** driver is organized into the following files:

tekpciPio.obj	Library file which contains all of the following routines.
tekpciPio.h	Header file with function prototypes for all external functions. This file should be #include'd by all user software that uses the **tekpciPio** driver.
tekpciPioIo.h	Header file with PIO module I/O definitions. This file is used by the driver routines and may be #include'd by user routines which need to perform direct I/O to PIO registers.
tekpciPioMain.c	Main driver file. Contains the following API functions (grouped by major function): Basic Functions: • tekpciPioDisplay • tekpciPioSetClockFreq • tekpciPioSetLocalBusClockFreq • tekpciPioDmaEnable • tekpciPioAttach • tekpciPioInit FPGA Functions: • tekpciPioFpgaReset • tekpciPioFpgaLookupId • tekpciPioFpgaIsLoaded • tekpciPioFpgaGetDwg • tekpciPioFpgaKill • tekpciPioFpgaLoad • tekpciPioFpgaLoadByDwg Parameter Functions:

- tekpciPioParmsDisplay
- tekpciPioRxGetParms
- tekpciPioRxSetParms
- tekpciPioTxGetParms
- tekpciPioTxSetParms

Receive Functions:

- tekpciPioRxGetCount
- tekpciPioRxGetPtr
- tekpciPioRxData
- tekpciPioRxFlush
- tekpciPioRxGetErrors
- tekpciPioRxClearErrors
- tekpciPioRxDmaStart
- tekpciPioRxDmaIsDone
- tekpciPioRxDmaQueue

Transmit Functions:

- tekpciPioTxGetCount
- tekpciPioTxGetFree
- tekpciPioTxGetPtr
- tekpciPioTxData
- tekpciPioTxGetErrors
- tekpciPioTxClearErrors
- tekpciPioTxDmaStart
- tekpciPioTxDmaIsDone
- tekpciPioTxDmaQueue

tekpciPioInfo.c — Routines to read and write EEPROM information about the PIO module. Contains the following API functions:

- tekpciPioInfoGet
- tekpciPioInfoSet
- tekpciPioInfoDisplay
- tekpciPioInfoInit
- tekpciPioInfoVerify

tekpciPioTest.c — Routines to perform confidence testing of the PIO module. Contains the following API functions:

- tekpciPioTestDatabus
- tekpciPioTestClock
- tekpciPioTestMemory

PIO Driver: Basic Functions

The **tekpciPio** basic functions, contained in **tekpciPioMain.c**, implement routines to attach a **TEKPCI_HANDLE** data structure to an PIO module and implement control and status functions which affect the PIO module as a whole (i.e. are not local to a specific RX or TX channel).

The **tekpciPio** basic functions are:

- **tekpciPioInit**, which performs global initialization of the **tekpciPio driver**.
- **tekpciPioAttach**, which attaches a **TEKPCI_HANDLE** data structure to an PIO module.
- **tekpciPioDisplay**, which displays control and status register information.
- **tekpciPioDmaEnable**, which enables or disables DMA requests.
- **tekpciPioSetClockFreq**, which is used to configure the ICD2053 clock synthesizer.
- **tekpciPioSetLocalBusClockFreq**, which is used to configure the local bus clock frequency between the PCI interface controller and the data link hardware.

Each of these functions is described in more detail below.

tekpciPioAttach( )

NAME

tekpciPioAttach( ) – Attach the **tekpciPio** driver to the specified handle.

SYNOPSIS

```
STATUS tekpciPioAttach (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine attempts to logically attach the **tekpciPio** driver to the *pci* handle. If the hardware is not accessible or not supported by the **tekpciPio** driver, the handle is not modified and ERROR is returned.

This routine performs basic initialization and fills in the required driver-specific fields of the **TEKPCI_HANDLE** structure. This routine must be called prior to using the *pci* handle with any other **tekpciPio** routines.

This routine typically calls an appropriate lower-level driver **Attach** routine for the PCI bus controller. For example, if the hardware module uses the PCI 9080 controller, this routine will call **tekpciPlx9080Attach( )** as a part of initializing the driver.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio

CAUTIONS

None

tekpciPioDisplay( )

NAME

tekpciPioDisplay( ) – Display control/status registers.

SYNOPSIS

```
STATUS tekpciPioDisplay (
    TEKPCI_HANDLE pci,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine reads the current control and status registers from the card specified by *pci* and displays a formatted status display using the user-supplied *output* and *handle* arguments.

The control and status registers are interpreted based on the FPGA identification value reported in the ID register. Unknown FPGA ID values are assumed to implement the baseline register set.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciPio driver**.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio

CAUTIONS

This routine is not aware of all FPGA implementations.

tekpciPioDmaEnable( )

NAME

tekpciPioDmaEnable( ) – Enable or disable DMA requests from the module.

SYNOPSIS

```
STATUS tekpciPioDmaEnable (
    TEKPCI_HANDLE pci,
    int enable
    );
```

DESCRIPTION

This routine enables or disables the DMA request signals from the hardware module specified by *pci* to the bus interface.

For modules which use the PCI 9080, the DMA controllers are integrated into the PCI 9080 interface and the module hardware is responsible for asserting or deasserting a DMA request when service is required. When DMA is disabled (*enable* is zero), the DMA requests are deasserted. When DMA is enabled, the DMA request signals follow the state of the associated module hardware.

DMA requests must typically be enabled prior to using the DMA controllers.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio

CAUTIONS

Disabling DMA requests is not recommended if there are any pending DMA operations.

tekpciPioInit( )

NAME

tekpciPioInit( ) – Perform global driver initialization.

SYNOPSIS

```
STATUS tekpciPioInit (void);
```

DESCRIPTION

This routine performs any required global initialization of the **tekpciPio** driver. It should be called prior to using any **tekpciPio** driver routines.

This routine should be called once after power-up initialization of the host environment, although it is not an error to call this routine multiple times.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if resources are not available for driver initialization.

SEE ALSO

tekpciPio

tekpciPioSetClockFreq( )

NAME

tekpciPioSetClockFreq( ) – Set clock synthesizer.

SYNOPSIS

```
STATUS tekpciPioSetClockFreq (
    TEKPCI_HANDLE pci,
    int channel,
    int *freq
    );
```

DESCRIPTION

This routine programs the ICD2053 clock synthesizer in the hardware module indicated by *pci* by calling **tekpciFlexSetClockFreq( )** at the lower level. The routine assumes that the FPGA is loaded and compliant.

The hardware module is assumed to have N clock synthesizers, with 0 … N-2 mapped to channels 0 to N-2 and clock synthesizer N-1 mapped to the local bus. If *channel* is –1, it maps to the local bus, otherwise, *channel* must be between 0 and N-1 where N is the number of non-local-bus synthesizers supported by the module.

The frequency with which the clock synthesizer is to be programmed is specified by the variable **freq*. **freq* is also modified by the routine to reflect the actual frequency finally programmed into the clock synthesizer.

If an invalid *channel* or **freq* is supplied, the routine will return ERROR. Additionally, if there is a problem accessing the hardware, the routine will also return ERROR. Otherwise it returns OK.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

ERROR if given an invalid parameter or there is a problem accessing hardware. OK otherwise.

SEE ALSO

tekpciPio, tekpciFlex, tekpciFlexSetClockFreq( )

tekpciPioSetLocalBusClockFreq( )

NAME

tekpciPioSetLocalBusClockFreq( ) – Set the local bus clock frequency.

SYNOPSIS

```
STATUS tekpciPioSetLocalBusClockFreq (
    TEKPCI_HANDLE pci,
    int *freq
    );
```

DESCRIPTION

This routine sets the local bus clock frequency for the hardware module specified by *pci*.

Some hardware modules have a fixed local bus clock, typically driven by the PCI bus clock. For those modules, this routine will return ERROR unless the user attempts to set the frequency to the correct value, in which case OK will be returned without any change to the hardware.

Other modules have a clock synthesizer to set the local bus clock frequency, in which case this routine sets the clock frequency to the value in **freq* (in Hertz). The value in **freq* is updated to reflect the actual clock synthesizer frequency based on the PLL programmed value. For arbitrary frequencies, the programmed PLL value will be within a small tolerance of the desired frequency.

The local bus clock frequency is usually based on the bus clock frequency. Deviations in the bus clock frequency will result in proportional deviations in the local bus clock frequency.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio

CAUTIONS

None

PIO Driver: FPGA Functions

The **tekpciPio** FPGA functions, contained in **tekpciPioMain.c** and **tekpciPioFpga.c**, implement routines to download FPGA images to the module, identify the FPGA image loaded into the module, and access standard FPGA images by calling functions in the **tekpciFlex** driver.

The full FPGA ID value consists of a five-digit drawing number, two-digit FPGA size, and two-digit revision number. For example, the standard streaming I/O FPGA for a Parallel ECL Input PMC module is drawing number 29421, FPGA size 50, and revision 01, making the ID value 294215001.

When looking up FPGA images, the revision field affects the search as follows: if the revision value is non-zero, the specific revision number is used; otherwise, the highest available revision is used. For example, if the driver has FPGA images 294215001 (rev -) and 294215002 (rev A) defined, the user application can use the rev - image by using an ID value of 294215001 or can use the latest available revision by using an ID value of 294215000.

If the user application has its own FPGA image to download, the user application can call the **tekpciPioFpgaLoad( )** routine with a pointer to the FPGA image (a **const TEK_FPGA_ENTRY** pointer). Typically, this requires that the user application include an FPGA image as a C language include file and build an appropriate **TEK_FPGA_ENTRY** data structure for the FPGA image.

If the user application needs to download a driver-supplied FPGA image, the user can first check if there is currently an FPGA loaded using **tekpciPioFpgaIsLoaded( )**. Then the user application might call **tekpciPioFpgaGetDwg( )** to check whether or not it is the desired FPGA and, if it is not, call **tekpciPioFpgaKill( )** in order to remove the FPGA image in preparation for a new one. If the user application knows the correct FPGA image, the user can then load the image using **tekpciPioFpgaLoad( )**, otherwise the user application can call the **tekpciPioFpgaLoadByDwg( )** routine as a shortcut.

The driver also provides the **tekpciPioFpgaLookupId( )** routine to map FPGA names into ID values for a specific PIO module. There is also **tekpciPioFpgaReset( )**, which resets the FPGA's internal logic.

tekpciPioFpgaGetDwg( )

NAME

tekpciPioFpgaGetDwg( ) – Get the FPGA drawing and revision information.

SYNOPSIS

```
STATUS tekpciPioFpgaGetDwg (
    TEKPCI_HANDLE pci,
    int *dwg,
    int *rev
    );
```

DESCRIPTION

This routine checks to see if an FPGA image is loaded into the PIO module specified by *pci* and then reads the FPGA drawing and revision information from the ID register in the local address space. This is done by calling **tekpciFlexFpgaGetDwg( )**.

If the FPGA is not loaded or if a hardware error is encountered, ERROR is returned. If the FPGA is loaded and the ID register is valid, OK is returned.

The drawing number portion of the ID register is copied into **dwg* and the revision number portion of the ID register is copied into **revision*. Either *dwg* or *revision* may be NULL, in which case no update is performed to that variable.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK if an FPGA is loaded, ERROR otherwise.

SEE ALSO

tekpciPio, tekpciPioIsFpgaLoaded( ), tekpciFlexFpgaGetDwg( )

tekpciPioFpgaIsLoaded( )

NAME

tekpciPioFpgaIsLoaded( ) – Check whether FPGA is loaded.

SYNOPSIS

```
STATUS tekpciPioFpgaIsLoaded (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine checks the FPGA device on the PIO module specified by *pci* which returns OK if the FPGA is loaded with a program image and ERROR otherwise.

The FPGA program image is detected by examining the FLEX 10K CONF_DONE signal. This will indicate whether a program image has been successfully downloaded; no checking is performed on whether the image is accessible through the local bus.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK if an FPGA is loaded, ERROR otherwise.

SEE ALSO

tekpciPio, tekpciPioFpgaLoad( )

tekpciPioFpgaKill( )

NAME

tekpciPioFpgaKill( ) – Reset FPGA image.

SYNOPSIS

```
STATUS tekpciPioFpgaKill (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine resets the FPGA image on the PIO module specified by *pci*. This causes the FPGA to exit user mode and await a new configuration image; all local bus operations will be invalid until a new image is downloaded using **tekpciPioFpgaLoad( )**.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioFpgaLoad( )

tekpciPioFpgaLoad( )

NAME

tekpciPioFpgaLoad( ) – Load an FPGA image.

SYNOPSIS

```
STATUS tekpciPioFpgaLoad (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    const TEK_FPGA_ENTRY *fep,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine loads the FPGA image specified by *fep* into the PIO module specified by *pci* and returns OK if the FPGA is loaded successfully and ERROR otherwise.

The *level* argument determines the amount of status display generated by the function. If *level* is **VERBOSE_NONE**, no status output is generated and output and handle may be NULL. If *level* is **VERBOSE_ERROR**, status output will be generated only in the case of an error. If *level* is **VERBOSE_ALL**, status output will be generated as the FPGA is downloaded.

All output messages are generated by calling *output* with *handle* as an argument.

INCLUDE FILES

tekpci.h, **tekpciPio.h, tekFpga.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK if an FPGA is loaded, ERROR otherwise.

SEE ALSO

tekpciPio, tekFpga, tekpciPioFpgaLoadByDwg( ), tekpciPioFpgaIsLoaded( )

tekpciPioFpgaLoadByDwg( )

NAME

tekpciPioFpgaLoadByDwg( ) – Lookup and then load an FPGA image by ID value.

SYNOPSIS

```
STATUS tekpciPioFpgaLoadByDwg (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    int dwg,
    int revision,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine looks up the FPGA image specified by *dwg* and *revision* and then, if the FPGA image is found, calls **tekpciPioFpgaLoad( )** to load the FPGA image into the PIO module specified by *pci*. If the lookup and load operations are both successful, OK is returned; otherwise, ERROR is returned.

The *dwg* argument is the drawing number of the desired FPGA image. The *dwg* value may be determined by the user or may be generated from an FPGA name by calling **tekpciPioFpgaLookupId( )**. The dwg and revision arguments are combined with the FPGA size on the PIO module to generate the FPGA image ID.

The *level* argument determines the amount of status display generated by the function. If *level* is **VERBOSE_NONE**, no status output is generated and output and handle may be NULL. If *level* is **VERBOSE_ERROR**, status output will be generated only in the case of an error. If *level* is **VERBOSE_ALL**, status output will be generated as the FPGA is downloaded.

All output messages are generated by calling *output* with *handle* as an argument.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK if an FPGA is loaded, ERROR otherwise.

SEE ALSO

tekpciPio, tekpciPioFpgaLoad( ), tekpciPioFpgaIsLoaded( )

tekpciPioFpgaLookupId( )

NAME

tekpciPioFpgaLookupId( ) – Lookup an FPGA ID by name.

SYNOPSIS

```
STATUS tekpciPioFpgaLookupId (
    TEKPCI_HANDLE pci,
    const char *name,
    int *id
    );
```

DESCRIPTION

This routine looks up the appropriate FPGA ID value for the specified FPGA function name as applied to the PIO module specified by *pci*.

The specific names are module-dependent. If name is NULL or "default", the most generic streaming I/O FPGA is identified.

The PIO module is used to determine the appropriate FPGA drawing and size based on the hardware revision level of the module and the FPGA size installed on the module.

If the name is invalid or if the required FPGA image is not accessible, ERROR is returned.

This routine does not actually load the FPGA image into the PIO module.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK if an FPGA image is available, ERROR otherwise.

SEE ALSO

tekpciPio, tekFpgaImageGet( )

tekpciPioFpgaReset( )

NAME

tekpciPioFpgaReset( ) – Reset FPGA internal logic.

SYNOPSIS

```
STATUS tekpciPioFpgaReset (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine resets the FPGA on the PIO module specified by *pci*. The FPGA reset is performed by writing the internal reset control bit. The semantics of the reset function are FPGA-dependent; the typical FPGA resets the RX and TX physical FIFOs along with all internal state machines and control registers.

The routine then enables the TX outputs, waits for a minimum of 1 millisecond, and then clears all pending error bits.

The user must have previously loaded an FPGA image to the card using the **tekpciPioFpgaLoad( )** or **tekpciPioFpgaLoadByDwg( )** routines.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioFpgaLoad( ), tekpciPioFpgaLoadByDwg( )

PIO Driver: Parameter Functions

The **tekpciPio driver** implements a set of functions which allow the user application to get and set interface parameters. Each channel has a separate parameter state structure which may be accessed.

The **tekpciPio** parameter API is intended to support several different hardware configurations with the possibility of different types of interactions between channels. In most cases, TEK's hardware products implement either one channel or, if multiple channels are implemented, independent channels. In cases where channel parameters are not independent, the parameter API will override previous settings with new settings.

If the user requires detailed knowledge about the hardware module's interactions, the documentation for the hardware module should be consulted. To verify proper parameter settings, the user application can set all desired parameter states for all transmit and receive channels and then read and verify all parameter states. If the hardware limitations resulted in the desired parameters being inconsistent, the resulting verify operation will fail, allowing the user application to detect a potential problem.

Channel parameters are accessed using a generic **TEKPCI_PIO_PARMS** parameter data structure which is defined in **tekpciPio.h**. The **tekpciPio** parameter API includes functions to get and set the parameter state for receive (RX) and transmit (TX) channels.

The **TEKPCI_PIO_PARMS** data structure will be expanded in future versions of the **tekpciPio** driver to support definition of additional hardware options. The existing field names and definitions will not be changed, although additional valid states may be added.

Most of the constant information in the **TEKPCI_PIO_PARMS** data structure is determined by the hardware model information that is maintained in the Serial EEPROM on the hardware module.

The **TEKPCI_PIO_PARMS** data structure consists of three portions. The first portion is a read-only set of fields that are filled in by the **GetParms** routines and ignored by the **SetParms** routines. The second set of fields are read-write fields which are filled in with the current hardware state by the **GetParms** routines and used to set the current hardware state by the **SetParms** routines. The third set of fields are informational fields which describe the hardware state after the parameter state has changed; these fields are filled in by the **GetParms** routines and are updated based on the programmed parameter set by the **SetParms** routines.

The recommended approach for changing the parameter state is to use **GetParms** to get the parameter state, modify the desired fields, and then use **SetParms** to set the new parameter state. This approach will automatically support future versions of the driver which define additional parameter fields.

The **TEKPCI_PIO_PARMS** data structure is shown below.

```
typedef struct tekpciPioParms {
    /* The readonly [RO] parameters describe the channel hardware */

    int numChannels;      /* Const: # of RX or TX channels */
    int rxCapable;        /* Const: NZ if channel is RX capable */
    int txCapable;        /* Const: NZ if channel is TX capable */
    int oscFreq;          /* Const: Oscillator freq (0 if none) */
    int synthMinFreq;     /* Const: Synthesizer min freq */
    int synthMaxFreq;     /* Const: Synthesizer max freq */
    int clkMaxFreq;       /* Const: Clock maximum frequency */
    int channelWidth;     /* Const: # of bits in physical interface */

    /* The readwrite [RW] parameters configure the hardware */

    int enable;           /* RW: Enable (RX or TX) */
    int outputEnable;     /* RW: NZ to enable output (TX only) */
    int rxClkSource;      /* RW: RX clock source (TTL or PECL)  */
    int txClkSource;      /* RW: TX clock source(OSC, SYNTH or BUSCLK)*/
    int synthReqFreq;     /* RW: Requested synthesizer freq */

    int enable;           /* NZ for enable (RX or TX) */
    int byteSwap;         /* NZ to swap bytes (RX or TX) */
    int rxDataValidIgnore; /* NZ to ignore DataValid (RX only) */
    int txClkSource;      /* RW: Transmit clk source (OSC, SYNTH or
                             BUSCLK) (TX only) */
    int synthReqFreq;     /* RW: Requested synthesizer freq */
    int wordSize;         /* RW: 8, 16 or 32 bits per output word */

    /* These parameters may change by operating mode and reflect the
current state of the hardware. */

    int synthActFreq;           /* RO: Actual synthesizer freq */
    int fifoSize;               /* RO: FIFO size in bytes */

} TEKPCI_PIO_PARMS;

#define TEKPCI_PIO_PARMS_TX_CLKSOURCE_SYNTH   0
#define TEKPCI_PIO_PARMS_TX_CLKSOURCE_BUSCLK  1
#define TEKPCI_PIO_PARMS_TX_CLKSOURCE_P2      2
```

For a given channel, the *oscFreq* field will be non-zero if the hardware module has a fixed frequency oscillator installed. The *synthMinFreq* and *synthMaxFreq* fields indicate the valid range of PLL-based synthesizer frequencies; the range is determined by the synthesizer device and is usually further limited by the selected interface. The field *clkMaxFreq* indicates the maximum clock frequency supported by the physical interface. The *channelWidth* field indicates the number of bits in the physical interface (typically 8, 16 or 32).

The *enable* field enables or disables the channel. For RX channels, disabling the channel has the effect of discarding all input data. For TX channels, disabling the channel has the effect of disabling the output drivers and suspending data transmission.

The *txClkSource* field specifies the source of the word rate clock, which can typically be a clock synthesizer or a fixed frequency oscillator. Again, the exact characteristics of the different clock sources are module-dependent.

The *synthReqFreq* field specifies the desired PLL clock synthesizer frequency, or more specifically the desired word rate for the channel. The synthesizer may be programmed through the **SetParms** routine even if the synthesizer is not selected as the clock source for the channel.

The *synthActFreq* field specifies the actual PLL clock synthesizer frequency assuming a perfect clock input. Typically, the PLL clock synthesizer can generate arbitrary clock frequencies with little or no error through appropriate programming of the synthesizer. However, the synthesizer programming parameters will often result in small fixed errors in frequency relative to the requested frequency. This field allows the user application to display the actual frequency and if necessary to ensure that the frequency is within the required tolerance for the user application. Errors in the reference input frequency (typically the module bus frequency) will result in linear errors to the actual frequency which are not reported through this field.

tekpciPioParmsDisplay( )

NAME

tekpciPioParmsDisplay( ) – Display the parameter state information.

SYNOPSIS

```
STATUS tekpciPioParmsDisplay (
    TEKPCI_PIO_PARMS *parms,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine displays a formatted version of the *parms* structure using the user-supplied *output* and *handle* arguments.

This routine does not access the hardware module; it is assumed that the info structure was filled in by a call to **tekpciPioGetParms( )**.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciPio driver**.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered with the *parms* data structure.

SEE ALSO

tekpciPio

tekpciPioRxGetParms( )

NAME

tekpciPioRxGetParms( ) – Get RX parameter state.

SYNOPSIS

```
STATUS tekpciPioRxGetParms (
    TEKPCI_HANDLE pci,
    int channel,
    TEKPCI_PIO_PARMS *parms
    );
```

DESCRIPTION

This routine gets the current parameter state for the specified RX channel on the specified *pci* hardware.

Channel must be a valid channel number for the hardware module referred to by *pci*.

The data structure at **parms* is filled in with the current parameter state for the receive channel.

The parameter state is typically determined by software data structures maintained as a part of the *pci* handle by the driver. If the user has directly accessed the hardware module, the reported parameter state may not be accurate.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if state in the TEKPCI handle doesn't match the hardware.

SEE ALSO

tekpciPio, tekpciPioRxSetParms( )

tekpciPioRxSetParms( )

NAME

tekpciPioRxSetParms( ) – Set RX parameter state and update hardware module.

SYNOPSIS

```
STATUS tekpciPioRxSetParms (
    TEKPCI_HANDLE pci,
    int channel,
    TEKPCI_PIO_PARMS *parms
    );
```

DESCRIPTION

This routine sets the current parameter state for the specified RX channel on the specified *pci* hardware and updates the hardware control registers accordingly.

Channel must be a valid channel number for the hardware module referred to by *pci*.

The data structure at **parms* is validated and then used to determine the new parameter state. Parameter fields in the *parms data structure which are negative are first filled in with the current parameter state.

The driver currently will not minimize hardware accesses, all hardware accesses will be performed regardless of whether the state has changed or not.

Updating the operating parameters will typically result in spurious data being received as the hardware is initialized to the new state. In particular, receive error status flags may be set as the hardware transitions from the old to the new state.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered.

SEE ALSO

tekpciPio, tekpciPioRxGetParms( )

tekpciPioTxGetParms( )

NAME

tekpciPioTxGetParms( ) – Get TX parameter state.

SYNOPSIS

```
STATUS tekpciPioTxGetParms (
    TEKPCI_HANDLE pci,
    int channel,
    TEKPCI_PIO_PARMS *parms
    );
```

DESCRIPTION

This routine gets the current parameter state for the specified TX channel on the specified *pci* hardware.

Channel must be a valid channel number for the hardware module referred to by *pci*.

The data structure at **parms* is filled in with the current parameter state for the receive channel.

The parameter state is typically determined by software data structures maintained as a part of the *pci* handle by the driver. If the user has directly accessed the hardware module, the reported parameter state may not be accurate.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if state in the TEKPCI handle doesn't match the hardware.

SEE ALSO

tekpciPio, tekpciPioTxSetParms( )

tekpciPioTxSetParms( )

NAME

tekpciPioTxSetParms( ) – Set TX parameter state and update hardware module.

SYNOPSIS

```
STATUS tekpciPioTxSetParms (
    TEKPCI_HANDLE pci,
    int channel,
    TEKPCI_PIO_PARMS *parms
    );
```

DESCRIPTION

This routine sets the current parameter state for the specified TX channel on the specified *pci* hardware and updates the hardware control registers accordingly.

Channel must be a valid channel number for the hardware module referred to by *pci*.

The data structure at **parms* is validated and then used to determine the new parameter state. Parameter fields in the *parms data structure which are negative are first filled in with the current parameter state.

The driver currently will not minimize hardware accesses, all hardware accesses will be performed regardless of whether the state has changed or not.

Updating the operating parameters will typically result in spurious data being received as the hardware is initialized to the new state. In particular, receive error status flags may be set as the hardware transitions from the old to the new state.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered.

SEE ALSO

tekpciPio, tekpciPioTxGetParms( )

PIO Driver: Receive Functions

The **tekpciPio** Receive functions provide the interface between the user application and the receive data stream from the hardware module. Using the Receive functions, the user application can get and clear error status conditions and transfer data from the module using either software memory accesses or DMA transfers.

The receive API consists of the following functions:

- **tekpciPioRxGetPtr( )** and **tekpciPioRxGetCount( )** allow the user application to access a direct FIFO interface to the receive data stream.
- **tekpciPioRxData( )** allows the user application to retrieve data from the RX FIFO.
- **tekpciPioRxDmaStart( )** and **tekpciPioRxDmaIsDone( )** allow the user application to perform DMA transfers from the receive data stream into user-specified buffer memory. These routines are not recommended for new designs; **tekpciPioRxDmaQueue( )** should be used instead.
- **tekpciPioRxDmaQueue( )** allows the user application to queue one or more DMA transfers and get callback notification when each DMA transfer is completed.
- **tekpciPioRxGetErrors( )**, which gets the errors on the RX channel.
- **tekpciPioRxClearErrors( )**, which resets the error status bits on the RX channel.

The Receive functions are implemented in **tekpciPioMain.c**.

tekpciPioRxClearErrors( )

NAME

tekpciPioRxClearErrors( ) – Reset RX error conditions.

SYNOPSIS

```
STATUS tekpciPioRxClearErrors (
    TEKPCI_HANDLE pci,
    int channel,
    int errors
    );
```

DESCRIPTION

This routine resets the RX error status bits on the hardware module. The hardware module is specified by *pci* and the receive channel is specified by *channel*.

Errors specifies the status bits to be reset. If *errors* is zero, all status bits are reset.

The hardware module typically implements receive errors as "sticky" status bits which are set when an error occurs and reset under software control. The **tekpciPioRxGetErrors( )** routine may be used to detect error conditions, after which **tekpciPioRxClearErrors( )** may be used to reset the errors which have been detected.

The definitions of specific error conditions are module-dependent. The API defines two common error types: **TEKPCI_PIO_ERROR_FIFO**, which indicates a low-level receive FIFO overflow, and **TEKPCI_PIO_ERROR_DATA**, which indicates a data quality error at the input. Additional error types may be defined by specific hardware or FPGA applications.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioRxGetErrors( )

tekpciPioRxData( )

NAME

TekpciPioRxData( ) – Get data in RX FIFO.

SYNOPSIS

```
STATUS tekpciPioRxGetCount (
    TEKPCI_HANDLE pci,
    int channel,
    int timeout,
    int *count,
    void *buf
    );
```

DESCRIPTION

This routine reads up to **count* words of data from the RX FIFO on the channel indicated by *channel* of the hardware module specified by *pci*. The read data is then stored using **buf*.

If *timeout* is zero, the routine will wait until all **count* words are available. If *timeout* is nonzero, the routine will wait for the requisite number of timer ticks. **Count* is then changed to reflect the actual number of words read.

The receive FIFO interface should not be used if DMA processing is being used for the channel.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

ERROR if there is a problem accessing hardware. OK otherwise.

SEE ALSO

tekpciPio

tekpciPioRxDmaIsDone( )

NAME

tekpciPioRxDmaIsDone( ) – Check for completion of an RX DMA transfer.

SYNOPSIS

```
STATUS tekpciPioRxDmaIsDone (
    TEKPCI_HANDLE pci,
    int channel
    );
```

DESCRIPTION

This routine checks to see if a pending Receive DMA transfer has been completed. If a Receive DMA transfer has been started and completed, OK is returned.

The hardware module is specified by *pci* and the receive channel is specified by *channel*.

The implementation of this function is module-dependent. For hardware modules which use the PCI 9080 interface controller, this routine checks both the PCI 9080 DMA controller "done" status bit and the internal software state which is updated by the **tekpciPlx9080** interrupt service procedure. Proper operation of this routine requires that the **tekpciPlx9080** interrupt handler is used for DMA completion interrupts.

Note that this routine will return ERROR if no Receive DMA transfer has ever been queued. This behavior may be changed in future releases of the driver.

This routine is only meaningful for direct DMA transfers which are queued with **tekpciPioRxDmaStart( )**. PIO-list DMA transfers are not supported by this API.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK if a Receive DMA transfer has been started and completed, or ERROR otherwise.

SEE ALSO

tekpciPio, tekpciPioRxDmaStart( )

tekpciPioRxDmaQueue( )

NAME

tekpciPioRxDmaQueue( ) – Queue a RX DMA transfer.

SYNOPSIS

```
STATUS tekpciPioRxDmaQueue (
    TEKPCI_HANDLE pci,
    int channel,
    UINT32 pciAddr,
    int length,
    void (*callback) (void *handle),
    void *handle
    );
```

DESCRIPTION

This routine queues a Receive DMA transfer to the receive channel specified by *channel* on the hardware module specified by *pci* to PCI target memory. The DMA transfer is performed to *pciAddr*, which is required to be a valid address in the PCI address space. A maximum of *length* bytes are transferred.

The user application is responsible for ensuring that *pciAddr* is a valid PCI address. Note that an address of a memory buffer in the host address space is not always the same as the PCI address for some host environments.

This routine transparently supports linked-list transfers. If the DMA channel is busy, the DMA transfer will be queued and executed when all pending operations have completed. If the DMA channel is idle, the DMA transfer will start immediately.

When the DMA transfer has completed, the user-supplied *callback* function will be called with the user-supplied *handle* argument. The callback may be executed in interrupt context, which may limit the types of operations the user routine may perform. If no callback routine is required, the *callback* argument should be NULL.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio

tekpciPioRxDmaStart( )

NAME

tekpciPioRxDmaStart( ) – Start a RX DMA transfer.

SYNOPSIS

```
STATUS tekpciPioRxDmaIsDone (
    TEKPCI_HANDLE pci,
    int channel,
    void *pciaddr,
    int length
    );
```

DESCRIPTION

This routine sets up a Receive DMA transfer from the receive channel specified by *channel* on the hardware module specified by *pci* to PCI target memory. The DMA transfer is performed to *pciaddr*, which is required to be a valid address in the PCI address space. A maximum of *length* bytes are transferred.

The user application is responsible for ensuring that *pciaddr* is a valid PCI address. Note that an address of a memory buffer in the host address space is not always the same as the PCI address for some host environments.

For raw I/O streams, the DMA transfer will always consist of *length* bytes. For some hardware modules and FPGA applications, the hardware may optionally terminate a DMA transfer prior to the end of the buffer. In this case, the length transferred may be less than or equal to the maximum buffer length. DMA transfers will never be longer than *length* bytes.

This routine only supports direct DMA transfers (i.e. linked-list DMA packets are not supported). The user application is required to ensure that the DMA controller is idle before initiating a DMA transfer using this routine. The state of the DMA controller can be checked using **tekpciPioRxDmaIsDone( )**.

INCLUDE FILES **tekpci.h**, **tekpciPio.h**

SOURCE FILES **tekpciPioMain.c**

RETURNS OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO **tekpciPio, tekpciPioRxDmaIsDone( )**

tekpciPioRxGetCount( )

NAME

tekpciPioRxGetCount( ) – Get current RX FIFO count.

SYNOPSIS

```
STATUS tekpciPioRxGetCount (
    TEKPCI_HANDLE pci,
    int channel,
    int *count
    );
```

DESCRIPTION

This routine returns the number of 32-bit words in the receive FIFO for the receive channel specified by *channel* in the hardware module specified by *pci*. The return value is written into **count*.

The user application is guaranteed that, after this function is called, at least **count* words may be read using the FIFO pointer returned by **tekpciPioRxGetPtr( )**.

This routine is intended for use with hardware modules and FPGA applications which implement raw streaming I/O. Applications which implement packet or message based I/O should generally not use this routine.

The receive FIFO interface should not be used if DMA processing is being used for the channel.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioRxGetPtr( )

tekpciPioRxGetErrors( )

NAME

tekpciPioRxGetErrors( ) – Get RX error conditions.

SYNOPSIS

```
STATUS tekpciPioRxGetErrors (
    TEKPCI_HANDLE pci,
    int channel,
    int *errors
    );
```

DESCRIPTION

This routine reads the current error status for the receive channel specified by *channel* on the *pci* hardware module.

The hardware module typically implements these errors as “sticky” status bits which are set when an error occurs and reset under software control. Errors which are reported by **tekpciPioRxGetErrors( )** may be cleared by calling the **tekpciPioRxClearErrors( )** routine.

The exact semantics of the error conditions are module-dependent. The API defines two common error types: **TEKPCI_PIO_ERROR_FIFO**, which indicates a low-level receive FIFO overflow, and **TEKPCI_PIO_ERROR_DATA**, which indicates a data quality error at the input. Additional error types may be defined by specific hardware or FPGA applications.

The error bits that are set are reported in **errors*. Errors are not cleared by this function; typically, errors will remain set until explicitly cleared.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioRxClearErrors( )

tekpciPioRxGetPtr( )

NAME

tekpciPioRxGetPtr( ) – Get pointer to RX FIFO data register.

SYNOPSIS

```
STATUS tekpciPioRxGetPtr (
    TEKPCI_HANDLE pci,
    int channel,
    volatile WORD32 **ptr
    );
```

DESCRIPTION

This routine returns a pointer to the receive FIFO data register for the receive channel specified by *channel* in the *pci* hardware module. The address of the receive FIFO in the host processor's address space is written into **ptr*.

This routine returns the FIFO pointer which may be used by the user application to read receive data. The number of words that may be safely read by the user application using this pointer may be determined by calling **tekpciPioRxGetCount( )**.

This routine is intended for use with hardware modules and FPGA applications which implement raw streaming I/O. Applications which implement packet or message based I/O should generally not use this routine.

The receive FIFO interface should not be used if DMA processing is being used for the channel.

The receive FIFO address is fixed once the hardware module has been loaded, the FPGA has been initialized, and the **tekpciPioRxSetParms( )** routine has been called.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioRxGetCount( )

PIO Driver: Transmit Functions

The **tekpciPio** Transmit functions provide the interface between the user application and the transmit data stream to the hardware module. Using the Transmit functions, the user application can get and clear error status conditions and transfer data to the module using either software memory accesses or DMA transfers.

The Transmit API consists of the following functions:

- **tekpciPioTxGetPtr( ), tekpciPioTxGetCount( )** and **tekpciPioTxGetFree( )** allow the user application to access a direct FIFO interface to the transmit data stream.
- **tekpciPioTxData( )** allows the user application to supply the TX FIFO with data.
- **tekpciPioTxDmaStart( )** and **tekpciPioTxDmaIsDone( )** allow the user application to perform DMA transfers from user-specified buffer memory into the transmit data stream. These routines are not recommended for new designs; **tekpciPioTxDmaQueue( )** should be used instead.
- **tekpciPioTxDmaQueue( )** allows the user application to queue one or more DMA transfers and get callback notification when each DMA transfer is completed.
- **tekpciPioTxGetErrors( )**, which gets the errors on the TX channel.
- **tekpciPioTxClearErrors( )**, which resets the error status bits on the TX channel.

The Transmit functions are implemented in **tekpciPioMain.c**.

tekpciPioTxClearErrors( )

NAME

tekpciPioTxClearErrors( ) – Reset TX error conditions.

SYNOPSIS

```
STATUS tekpciPioTxClearErrors (
    TEKPCI_HANDLE pci,
    int channel,
    int errors
    );
```

DESCRIPTION

This routine resets the TX error status bits on the hardware module. The hardware module is specified by *pci* and the transmit channel is specified by *channel*.

Errors specifies the status bits to be reset. If *errors* is zero, all status bits are reset.

The hardware module typically implements receive errors as "sticky" status bits which are set when an error occurs and reset under software control. The **tekpciPioTxGetErrors( )** routine may be used to detect error conditions, after which **tekpciPioTxClearErrors( )** may be used to reset the errors which have been detected.

The definitions of specific error conditions are module-dependent. The API defines one common error type: **TEKPCI_PIO_ERROR_FIFO**, which indicates a low-level transmit FIFO overflow. Additional error types may be defined by specific hardware or FPGA applications.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioTxGetErrors( )

tekpciPioTxData( )

NAME

tekpciPioTxData( ) – Write data to TX FIFO.

SYNOPSIS

```
STATUS tekpciPioTxData (
    TEKPCI_HANDLE pci,
    int channel,
    int timeout,
    int *count,
    void *buf
    );
```

DESCRIPTION

This routine writes up to **count* 32-bit words to the transmit FIFO of the channel on the hardware module specified by the user-supplied variables *channel* and *pci* respectively. The words to be written to the FIFO are passed to the routine using *buf*.

If *timeout* is zero, the routine will wait for all **count* words to be written to the transmit FIFO. If *timeout* is nonzero, then the routine writes until the requisite number of timer tickets have passed. Once writing has been completed, **count* is modified to reflect the actual number of words written to the FIFO.

The transmit FIFO interface should not be used if DMA processing is being used for the channel.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio

tekpciPioTxDmaIsDone( )

NAME

tekpciPioTxDmaIsDone( ) – Check for completion of a TX DMA transfer.

SYNOPSIS

```
STATUS tekpciPioTxDmaIsDone (
    TEKPCI_HANDLE pci,
    int channel
    );
```

DESCRIPTION

This routine checks to see if a pending Transmit DMA transfer has been completed. If a Transmit DMA transfer has been started and completed, OK is returned.

The hardware module is specified by *pci* and the receive channel is specified by *channel*.

The implementation of this function is module-dependent. For hardware modules which use the PCI 9080 interface controller, this routine checks both the PCI 9080 DMA controller "done" status bit and the internal software state which is updated by the **tekpciPlx9080** interrupt service procedure. Proper operation of this routine requires that the **tekpciPlx9080** interrupt handler is used for DMA completion interrupts.

Note that this routine will return ERROR if no Transmit DMA transfer has ever been queued. This behavior may be changed in future releases of the driver.

This routine is only meaningful for direct DMA transfers which are queued with **tekpciPioTxDmaStart( )**. Linked-list DMA transfers are not supported by this API.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK if a Transmit DMA transfer has been started and completed, or ERROR otherwise.

SEE ALSO

tekpciPio, tekpciPioTxDmaStart( )

tekpciPioTxDmaQueue( )

NAME

tekpciPioTxDmaQueue( ) – Queue a TX DMA transfer.

SYNOPSIS

```
STATUS tekpciPioTxDmaQueue (
    TEKPCI_HANDLE pci,
    int channel,
    UINT32 pciAddr,
    int length,
    void (*callback) (void *handle),
    void *handle
    );
```

DESCRIPTION

This routine queues a Transmit DMA transfer to the transmit channel specified by *channel* on the hardware module specified by *pci* to PCI target memory. The DMA transfer is performed from *pciAddr*, which is required to be a valid address in the PCI address space. A maximum of *length* bytes are transferred.

The user application is responsible for ensuring that *pciAddr* is a valid PCI address. Note that an address of a memory buffer in the host address space is not always the same as the PCI address for some host environments.

This routine transparently supports linked-list transfers. If the DMA channel is busy, the DMA transfer will be queued and executed when all pending operations have completed. If the DMA channel is idle, the DMA transfer will start immediately.

When the DMA transfer has completed, the user-supplied *callback* function will be called with the user-supplied *handle* argument. The callback may be executed in interrupt context, which may limit the types of operations the user routine may perform. If no callback routine is required, the *callback* argument should be NULL.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio

tekpciPioTxDmaStart( )

NAME

tekpciPioTxDmaStart( ) – Start a TX DMA transfer.

SYNOPSIS

```
STATUS tekpciPioTxDmaIsDone (
    TEKPCI_HANDLE pci,
    int channel,
    void *pciaddr,
    int length
    );
```

DESCRIPTION

This routine sets up a Transmit DMA transfer to the transmit channel specified by *channel* on the hardware module specified by *pci* to PCI target memory. The DMA transfer is performed from *pciaddr*, which is required to be a valid address in the PCI address space. A maximum of *length* bytes are transferred.

The user application is responsible for ensuring that *pciaddr* is a valid PCI address. Note that an address of a memory buffer in the host address space is not always the same as the PCI address for some host environments.

For raw I/O streams, the DMA transfer will always consist of *length* bytes. For some hardware modules and FPGA applications, the hardware may optionally terminate a DMA transfer prior to the end of the buffer, although this would be an unusual option for a transmit function. In this case, the length transferred may be less than or equal to the maximum buffer length. DMA transfers will never be longer than *length* bytes.

This routine only supports direct DMA transfers (i.e. linked-list DMA packets are not supported). The user application is required to ensure that the DMA controller is idle before initiating a DMA transfer using this routine. The DMA controller state can be checked using **tekpciPioTxDmaIsDone( )**.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioTxDmaIsDone( )

tekpciPioTxGetCount( )

NAME

tekpciPioTxGetCount( ) – Get current TX FIFO count.

SYNOPSIS

```
STATUS tekpciPioTxGetCount (
    TEKPCI_HANDLE pci,
    int channel,
    int *count
    );
```

DESCRIPTION

This routine returns the number of 32-bit words in the transmit FIFO for the transmit channel specified by *channel* in the hardware module specified by *pci*. The return value is written into **count*.

This routine is provided for diagnostic purposes. If the user application needs to write data to the TX FIFO, the **tekpciPioTxGetFree( )** routine should be used to determine the amount of free space available.

This routine is intended for use with hardware modules and FPGA applications which implement raw streaming I/O. Applications which implement packet or message based I/O should generally not use this routine.

The transmit FIFO interface should not be used if DMA processing is being used for the channel.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioTxGetPtr( )

tekpciPioTxGetErrors( )

NAME

tekpciPioTxGetErrors( ) – Get TX error conditions.

SYNOPSIS

```
STATUS tekpciPioTxGetErrors (
    TEKPCI_HANDLE pci,
    int channel,
    int *errors
    );
```

DESCRIPTION

This routine reads the current error status for the transmit channel specified by *channel* on the *pci* hardware module.

The hardware module typically implements these errors as “sticky” status bits which are set when an error occurs and reset under software control. Errors which are reported by **tekpciPioTxGetErrors( )** may be cleared by calling the **tekpciPioTxClearErrors( )** routine.

The exact semantics of the error conditions are module-dependent. The API defines one common error type: **TEKPCI_PIO_ERROR_FIFO**, which indicates a low-level transmit FIFO overflow. Additional error types may be defined by specific hardware or FPGA applications.

The error bits that are set are reported in **errors*. Errors are not cleared by this function; typically, errors will remain set until explicitly cleared.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioTxClearErrors( )

tekpciPioTxGetFree( )

NAME

tekpciPioTxGetFree( ) – Get current TX FIFO free space.

SYNOPSIS

```
STATUS tekpciPioTxGetFree (
    TEKPCI_HANDLE pci,
    int channel,
    int *count
    );
```

DESCRIPTION

This routine returns the number of 32-bit words of free space available in the transmit FIFO for the transmit channel specified by *channel* in the hardware module specified by *pci*. The return value is written into **count*.

The user application is guaranteed that, after this function is called, at least **count* words may be written using the FIFO pointer returned by **tekpciPioTxGetPtr( )**.

This routine is intended for use with hardware modules and FPGA applications that implement raw streaming I/O. Applications which implement packet or message based I/O should generally not use this routine.

The transmit FIFO interface should not be used if DMA processing is being used for the channel.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioTxGetPtr( )

tekpciPioTxGetPtr( )

NAME

tekpciPioTxGetPtr( ) – Get a pointer to the TX FIFO data register.

SYNOPSIS

```
STATUS tekpciPioTxGetPtr (
    TEKPCI_HANDLE pci,
    int channel,
    volatile WORD32 **ptr
    );
```

DESCRIPTION

This routine returns a pointer to the transmit FIFO data register for the transmit channel specified by *channel* in the *pci* hardware module. The address of the transmit FIFO in the host processor's address space is written into **ptr*.

This routine returns the FIFO pointer which may be used by the user application to send transmit data. The number of words that may be safely written by the user application using this pointer may be determined by calling **tekpciPioTxGetFree( )**.

This routine is intended for use with hardware modules and FPGA applications which implement raw streaming I/O. Applications which implement packet or message based I/O should generally not use this routine.

The transmit FIFO interface should not be used if DMA processing is being used for the channel.

The transmit FIFO address is fixed once the hardware module has been loaded, the FPGA has been initialized, and the **tekpciPioTxSetParms( )** routine has been called.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio, tekpciPioTxGetFree( )

PIO Driver: Info Functions

The **tekpciPio** driver implements a set of functions which allow the user application to read and display information about the type of hardware module installed. The **tekpciPioInfo** functions work with a **TEKPCI_PIO_INFO** data structure, which is defined in **tekpciPio.h**. The **TEKPCI_PIO_INFO** data structure is shown below.

```
typedef struct tekpciPioInfo {
    int model;
    int serialnum;            /* Typical: 1998390001 */
    int model_backend         /* Model # of back end (0 if monolithic) */
    int serialnum_backend;    /* Serial # of back end (0 if monolithic) */
    char modelname[80];
    char modelname_backend[80];
    int hw_revision;          /* Typical: 1 */
    int memory_size;          /* 0 or 1024 (in KB) */
    int fpga_family;          /* 0 = 10K, 1 = 10KA,
                                 2 = 10KV, 3 = 10KE */
    int fpga_size;            /* 30 = 10K30, 50 = 10K50 */
    int fpga_speed;           /* 1 to 3 = dash-1 to dash-3 */
} TEKPCI_PIO_INFO;
```

The **tekpciPioInfo** API provides functions to display, get, initialize, set, verify and update the above data structure. Except under special circumstances, user applications will typically only need the **tekpciPioInfoGet( )** and **tekpciPioInfoDisplay( )** routines.

The **TEKPCI_PIO_INFO** data structure will be expanded in future versions of the **tekpciPio** driver to support definition of additional hardware options. The existing field names and definitions will not be changed, although additional valid states may be added.

User applications that need to determine the valid parameters for interface channels (i.e. maximum word rate) are encouraged to use the **tekpciPio** parameter API calls (**tekpciPioRxGetParms( )**, **tekpciPioTxGetParms( )**) instead of the **tekpciPioInfo** routines.

Most of the information in the **TEKPCI_PIO_INFO** data structure is maintained in the Serial EEPROM on the hardware module. The **tekpciPioInfo** routines call the lower-level **tekpciPlx9080Eeprom** routines to access the Serial EEPROM data, but the data values are interpreted by the **tekpciPio** driver.

tekpciPioInfoDisplay( )

NAME

tekpciPioInfoDisplay( ) – Display the module information.

SYNOPSIS

```
STATUS tekpciPioInfoDisplay (
    TEKPCI_PIO_INFO *infop,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine displays a formatted version of the *infop* structure using the user-supplied *output* and *handle* arguments.

This routine does not access the hardware module; it is assumed that the info structure was filled in by a call to **tekpciPioInfoGet( )**.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciPio driver**.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioInfo.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio

tekpciPioInfoGet( )

NAME

tekpciPioInfoGet( ) – Read the module-specific capability data from the hardware module.

SYNOPSIS

```
STATUS tekpciPioInfoGet (
    TEKPCI_HANDLE pci,
    TEKPCI_PIO_INFO *infop
    );
```

DESCRIPTION

This routine reads the module-specific capability information from the Serial EEPROM on the *pci* hardware module and stores the module information at **infop*.

If the module has not been initialized with a valid Serial EEPROM image, including the PCI configuration space model number, ERROR is returned. In this case, **infop* will be filled with indeterminate data.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioInfo.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPio

tekpciPioInfoInit( )

NAME

tekpciPioInfoInit( ) – Write module-specific PCI configuration information to the hardware module.

SYNOPSIS

```
STATUS tekpciPioInfoInit (
    TEKPCI_HANDLE pci,
    int model,
    int submodel,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine writes the PCI configuration portion of the Serial EEPROM to the *pci* hardware module.

Model must be set to a valid TEK model number for a hardware module. *Model* is not checked against the hardware module; because this routine is used for initial configuration of a hardware module, it is required to operate on uninitialized modules which by definition have no embedded identification information.

The *output* and *handle* arguments are used to display error messages in the event of a write or verify error.

If the module was not previously initialized, it will typically be necessary to reboot the host processor to allow the PCI address spaces to be remapped.

This routine is globally-scoped to support TEK's manufacturing test and initialization routines. This routine should not generally be called by user applications.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioInfo.c

RETURNS

OK, or ERROR if an error is encountered writing or verifying the Serial EEPROM on the hardware module.

SEE ALSO

tekpciPio, tekpciPioInfoVerify( )

tekpciPioInfoSet( )

NAME

tekpciPioInfoSet( ) – Write module-specific capability data to the hardware module.

SYNOPSIS

```
STATUS tekpciPioInfoSet (
    TEKPCI_HANDLE pci,
    TEKPCI_PIO_INFO *infop,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine writes the module-specific capability information from the **infop* data structure to the Serial EEPROM on the *pci* hardware module.

The **infop* data structure is checked for valid parameter values within the numeric range of values which are supported by the internal data structures. This is a broader range of values than are actually possible for most hardware modules. Therefore, it is easily possible for this routine to program invalid parameters for a specific hardware module. For example, the FPGA speed parameter can support values between 0 and 15, but the only supported values with current hardware are between 1 and 3.

The *output* and *handle* arguments are used to display error messages in the event of a parameter, write or verify error.

The module should be initialized with **tekpciPioInfoInit( )** before calling this function.

This routine is globally-scoped to support TEK's manufacturing test and initialization routines. This routine should not generally be called by user applications.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioInfo.c

RETURNS

OK, or ERROR if an error is encountered with the input parameters or in writing or verifying the Serial EEPROM on the hardware module.

SEE ALSO

tekpciPio, tekpciPioInfoGet( )

tekpciPioInfoVerify( )

NAME

tekpciPioInfoVerify( ) – Verify the module-specific PCI configuration information in the hardware module.

SYNOPSIS

```
STATUS tekpciPioInfoVerify (
    TEKPCI_HANDLE pci,
    int model,
    int submodel,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine verifies the PCI configuration portion of the Serial EEPROM to the *pci* hardware module.

Model must be set to a valid TEK model number for a hardware module.

The *output* and *handle* arguments are used to display error messages in the event of a verify error.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioInfo.c

RETURNS

OK, or ERROR if an error is encountered writing or verifying the Serial EEPROM on the hardware module.

SEE ALSO

tekpciPio, tekpciPioInfoInit( )

PIO Driver: Test Functions

The **tekpciPio driver** implements Built-In-Test functions, which support module confidence testing. The Built-In-Test functions may be used by user applications to verify basic operation of the hardware module.

Typically, a confidence test of a module consists of the following steps:

- As a part of initialization, verify the configuration space for the module. Modules which do not have valid configuration information will typically not be mapped to the **tekpciPio** driver.
- Once the module has been attached to a driver, verify data bus functionality using the **tekpciPioTestDatabus( )** function.
- If internal manufacturing tests are desired, call the **tekpciPioTestClock( )** routine to perform internal testing of the module.

If a test failure occurs, it may be useful to perform the underlying **tekpciPlx9080** test routines to isolate the cause of the failure.

tekpciPioTestClock( )

NAME

tekpciPioTestClock( ) – Test the local clock synthesizer and oscillator functions.

SYNOPSIS

```
STATUS tekpciPioTestClock (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    int channel,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine performs a test of the *pci* hardware functions which implement clock generation for the specified *channel*. Onboard oscillators and clock synthesizers, to the extent they are programmable, are exercised. If the hardware module includes local bus clock capabilities, those capabilities are also exercised.

If the hardware uses customizable FPGA programs, this routine assumes that a standard FPGA has been loaded. The specific functions performed by this test are module-dependent. After this test completes, the onboard clock generator state is indeterminate.

If *level* is **VERBOSE_NONE**, no display output is generated and the *output* and *handle* arguments may be NULL. If level is **VERBOSE_ERROR**, display output is generated only in the event of an error. If level is **VERBOSE_ALL**, display output is generated to report the results of the test.

Messages are displayed by calling *(*output) ( )* using the supplied *handle* argument.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciPio driver**.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioTest.c

RETURNS

OK, or ERROR if an error is encountered.

SEE ALSO

tekpciPio

tekpciPioTestDatabus( )

NAME

tekpciPioTestDatabus( ) – Test the local data bus to the hardware.

SYNOPSIS

```
STATUS tekpciPioTestDatabus (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine first performs a test of the *pci* hardware module using the **tekpciPlx9080TestDatabus( )** routine and then performs testing of the local data bus between the PCI 9080 controller and the onboard control/status registers.

All current TEK PCI/PMC products use the PCI 9080 controller. If a future product uses a different PCI controller, this test API will remain unchanged but the API will call the appropriate underlying driver.

If *level* is **VERBOSE_NONE**, no display output is generated and the *output* and *handle* arguments may be NULL. If level is **VERBOSE_ERROR**, display output is generated only in the event of an error. If level is **VERBOSE_ALL**, display output is generated to report the results of the test.

Messages are displayed by calling *(*output) ( )* using the supplied *handle* argument.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciPio** driver.

INCLUDE FILES

tekpci.h, **tekpciPio.h**

SOURCE FILES

tekpciPioTest.c

RETURNS

OK, or ERROR if an error is encountered.

SEE ALSO

tekpciPio, tekpciPlx9080

PLX9080 Driver API

The **tekpciPlx9080** VxWorks software driver for the PLX Technologies PCI 9080 provides functions which initialize the PCI 9080 controller, read and write EEPROM memory, enable/disable interrupts, configure and enable DMA, and perform confidence testing.

These functions are typically invoked by a higher level driver and not by a user application.

The **tekpciPlx9080** driver is organized into the following files:

tekpciPlx9080.o	Library file which contains all of the following routines.
tekpciPlx9080.h	Header file with function prototypes for all external functions. This file should be #include'd by all user software that uses the **tekpciPlx9080** driver.
tekpciPlx9080Io.h	Header file with PCI 9080 I/O definitions. This file is used by the driver routines and may be #include'd by user routines which need to perform direct I/O to PCI 9080 registers.
tekpciPlx9080Main.c	Main driver file. Contains the following API functions: • tekpciPlx9080Attach • tekpciPlx9080GetUserIo • tekpciPlx9080SetBusRequest • tekpciPlx9080SetBusDescript • tekpciPlx9080IntDisplay • tekpciPlx9080IntEnable • tekpciPlx9080Reload • tekpciPlx9080SetUserIo
tekpciPlx9080Dma.c	Routines to manage DMA transfers using the PCI 9080 integrated DMA controllers. Contains the following API functions: • tekpciPlx9080DmaDisplay • tekpciPlx9080DmaStart • tekpciPlx9080DmaIsDone
tekpciPlx9080Eeprom.c	Routines to read and write EEPROM data. Contains the following API functions: • tekpciPlx9080EepromIsProtectEnabled • tekpciPlx9080EepromRead • tekpciPlx9080EepromReadN • tekpciPlx9080EepromWrite • tekpciPlx9080EepromWriteN

	• tekpciPlx9080EepromWriteEnable • tekpciPlx9080EepromWriteAll • tekpciPlx9080EepromProtectClear • tekpciPlx9080EepromProtectRead • tekpciPlx9080EepromProtectWrite
tekpciPlx9080Test.c	Routines to perform confidence testing of the PCI 9080. Contains the following API functions: • tekpciPlx9080TestRegister • tekpciPlx9080TestDatabus

The following sections define the various global functions in the **tekpciPlx9080** driver. Functions are organized by file and major function.

PLX9080 Driver: Basic Functions

The **tekpciPlx9080** basic functions, contained in **tekpciPlx9080Main.c**, implement routines to attach a TEKPCI handle to a PCI 9080 device and access low-level PCI 9080 control and status signals.

tekpciPlx9080Attach( )

NAME

tekpciPlx9080Attach( ) – Attach the TEKPCI handle to the PCI 9080 driver.

SYNOPSIS

```
STATUS tekpciPlx9080Attach (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine initializes the *pci* data structure for operation with a PCI 9080 controller.

This routine is typically called by a higher-level driver as a part of the driver's **Attach** routine. For example, if a hardware module is a PIO PMC, the **tekpciPioAttach( )** function first calls **tekpciPlx9080Attach( )** and then attaches the higher-level **tekpciPio driver**.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Main.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080

tekpciPlx9080GetUserIo( )

NAME

tekpciPlx9080GetUserIo( ) – Get PCI 9080 input status.

SYNOPSIS

```
STATUS tekpciPlx9080GetUserIo (
    TEKPCI_HANDLE pci,
    int *eedata,
    int *useri
    );
```

DESCRIPTION

This routine reads the PCI 9080 CNTRL register and reports the value of the EEDI and USERI input signals.

If the EEDI input is asserted (high), **eedata* is set to a non-zero value; otherwise, **eedata* is cleared. If the USERI input is asserted (high), **useri* is set to a non-zero value; otherwise, it is cleared. Both *eedata* and *useri* may be NULL if no update is desired.

The EEDI signal is typically used to communicate with the Serial EEPROM. In some modules, the EEDI signal is used for an alternate function when the Serial EEPROM is not selected. The user application must ensure that the User I/O signals are not modified simultaneously by more than one task or erroneous operation may result.

This function is globally scoped to allow higher level drivers to access the PCI 9080 I/O signals. Generally, user applications should call the appropriate module-specific API to access hardware features of the card.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Main.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080, tekpciPlx9080SetUserIo( )

tekpciPlx9080SetBusRequest( )

NAME

tekpciPlx9080SetBusRequest( ) – Enables/disables the local bus request input.

SYNOPSIS

```
STATUS tekpciPlx9080SetBusRequest (
    TEKPCI_HANDLE pci,
    int *enable
    );
```

DESCRIPTION

This routine sets the local bus request input of the module specified by *pci* depending on *enable*. If *enable* is set, it enables the bus. Otherwise it disables it.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Main.c

RETURNS

OK

SEE ALSO

tekpciPlx9080

tekpciPlx9080SetBusDescript( )

NAME

tekpciPlx9080SetBusDescript( ) – Sets local bus descriptor for a specific address space.

SYNOPSIS

```
STATUS tekpciPlx9080SetBusDescript (
    TEKPCI_HANDLE pci,
    int space,
    int ready,
    int bterm
    );
```

DESCRIPTION

This routine configures the local bus descriptor indicated by *pci* for a specific bus address as specified by *space*. The local bus descriptor is updated to reflect the user-supplied *ready* and *bterm*.

Space is used to distinguish which address space to set. If *space* is 0, it sets the local bus descriptor for local address space 0. If *space* is 1, it sets the descriptor for local address space 1. If *space* is 2, it sets the descriptor for expansion rom space. If *space* is –1, the routine will set the local bus descriptors for all address spaces.

If the user-supplied *space* is not between –1 and 2, the routine will return ERROR. Otherwise it returns OK

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Main.c

RETURNS

OK, or ERROR if given an invalid argument.

SEE ALSO

tekpciPlx9080

tekpciPlx9080SetUserIo( )

NAME

tekpciPlx9080SetUserIo( ) – Set PCI 9080 control outputs.

SYNOPSIS

```
STATUS tekpciPlx9080SetUserIo (
    TEKPCI_HANDLE pci,
    int eecs,
    int eedata,
    int eeclk,
    int usero
    );
```

DESCRIPTION

This routine updates the PCI 9080 CNTRL register with the specified states for the EEPROM chip select (EECS), EEPROM data output (EEDATA), EEPROM clock (EECLK) and User Output (USERO) values.

For each of *eecs*, *eedata*, *eeclk* and *usero*, the corresponding control output is unmodified if the argument is negative, cleared if the argument is zero, and set if the argument is positive and non-zero.

The EECS, EEDATA and EECLK signals are also used to communicate with the Serial EEPROM. The user application should ensure that the PCI 9080 control signals are not modified simultaneously by more than one task or erroneous operation may result.

This function is globally scoped to allow higher level drivers to access the PCI 9080 I/O signals. Generally, user applications should call the appropriate module-specific API to access hardware features of the card.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Main.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080, tekpciPlx9080GetUserIo( )

tekpciPlx9080Reload( )

NAME

tekpciPlx9080Reload( ) – Reload PCI 9080 registers from Serial EEPROM.

SYNOPSIS

```
STATUS tekpciPlx9080Reload (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine initiates a reload of the PCI 9080 registers from the Serial EEPROM by setting the Reload Configuration bit in the PCI 9080 CNTRL register. All assigned address resources are lost.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Main.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080

PLX9080 Driver: Interrupt Functions

The **tekpciPlx9080** interrupt functions, contained in **tekpciPlx9080Main.c**, implement routines to enable/disable local interrupts and display interrupt statistics. The driver also implements an interrupt handler which may be "hooked" by the application into the appropriate PCI interrupt chain.

The **tekpciPlx9080** driver is designed to implement all of the 9080-specific functions for a PCI/PMC module. Generally, the services provided by the **tekpciPlx9080** driver are used by a higher-level driver to implement module-specific functions. The **tekpciPlx9080** interrupt and DMA functions are not typically called directly by the user application.

The relationship between the user application, top-level driver and tekpciPlx9080 driver are shown in the diagram below.

The **tekpciPlx9080** interrupt service procedure (ISP) is used to handle PCI interrupts which are generated by the PCI 9080. The driver ISP is designed to handle two types of interrupts:

- DMA completion interrupts are handled internally by the driver. If the direct DMA mode is being used, the DMA completion interrupt simply updates state variables which are contained in the TEKPCI handle. The state variables are used by the **tekpciPlx9080DmaIsDone( )** routine to determine whether a DMA operation has been completed.

Future versions of the **tekpciPlx9080** driver are expected to fully implement queued DMA packets. If linked-list DMA mode is being used, the ISP will update internal state variables as required to maintain a linked-list of DMA packets and to implement callback routines to higher level functions.

- Local module interrupts are acknowledged by the ISP and a callback function is executed. Typically, the callback function is installed by a higher level driver to perform module-specific functions. For example, if an PIO module implements a packet-based protocol, the local interrupt may be used for end-of-packet events and the callback function allows the PIO driver to perform end-of-packet processing.

The driver ISP

The TEKPCI handle structure defines two ISP routines which the **tekpciPlx9080** driver implements:

```
void (*isp_void) (TEKPCI_HANDLE pci);
STATUS (*isp_status) (TEKPCI_HANDLE pci);
```

The **tekpciPlx9080Attach( )** routine installs pointers to local **tekpciPlx9080** ISP routines for both the *isp_void* and *isp_status* fields in the TEKPCI handle.

These routines are intended to be “hooked” into the user application’s PCI interrupt chain for the appropriate interrupt vector. The PCI bus uses shared interrupt requests, which requires that each device which is capable of requesting interrupts implement a polling ISP routine. The device’s polling ISP routine is required to check to see if the device is requesting service and performs appropriate action only if an interrupt request is active. Because the chain of polling ISP routines is called for any device sharing the interrupt request, the driver’s polling ISP routine must be able to handle being called when the hardware does not require service.

The two ISP functions are provided to allow the driver to handle different types of Board Support Package (BSP) calling conventions. Some BSPs always execute all “hooked” ISP routines for any interrupt; these BSPs tend to require a void-returning ISP function as no status information is required. Other BSPs abort interrupt processing once an ISP routine indicates that an interrupt source has been handled; these BSPs use the STATUS-returning ISP to detect whether the driver detected and handled an interrupt. Both routines perform identical functions; the *isp_void* routine simply calls the *isp_status* routine and then discards the return status.

The internal ISP routine does the following:

- If the PCI 9080 DMA controller #0 is asserting an interrupt, the internal DMA state is updated and the interrupt cleared.
- If the PCI 9080 DMA controller #1 is asserting an interrupt, the internal DMA state is updated and the interrupt cleared.
- If the local interrupt is asserted and the *pci->local_isp* routine is defined, the *pci->local_isp* routine is called. If the local interrupt is still asserted, the local interrupt enable is cleared to prevent continuous interrupts.

The internal ISP also maintains a number of ISP counters for diagnostic purposes. The counters are maintained as a part of the TEKPCI handle structure and may be displayed with **tekpciPlx9080IntDisplay( )**.

tekpciPlx9080IntDisplay( )

NAME

tekpciPlx9080IntDisplay( ) – Display interrupt registers.

SYNOPSIS

```
STATUS tekpciPlx9080IntDisplay (
    TEKPCI_HANDLE pci,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine displays the Interrupt Service Procedure (ISP) counters and PCI 9080 INTCSR register for the card specified by *pci* and displays a formatted status display using the user-supplied *output* and *handle* arguments.

The ISP counters are maintained only if the driver's ISP routine is used to handle PCI interrupts. The ISP maintains four counts: (1) the number of times the ISP has been invoked; (2) the number of times the ISP has been invoked with no interrupt asserted; (3) the number of times the ISP has been invoked with the PCI 9080 local interrupt asserted; and (4) the number of times that the ISP was invoked with the local interrupt asserted when calling the ISP callback routine did not clear the local interrupt.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciPlx9080** driver.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Main.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080

tekpciPlx9080IntEnable( )

NAME

tekpciPlx9080IntEnable( ) – Enable/disable PCI 9080 local interrupts.

SYNOPSIS

```
STATUS tekpciPlx9080IntEnable (
    TEKPCI_HANDLE pci,
    int enable
    );
```

DESCRIPTION

This routine enables or disables PCI 9080 local interrupts. The specific function implemented through local interrupts is module-dependent.

If *enable* is non-zero, local interrupts are enabled; otherwise, local interrupts are disabled.

This function does not affect interrupts generated by the PCI 9080 DMA controllers.

The driver assumes that the user application has hooked either *pci->isp_void( )* or *pci->isp_status( )* into the user interrupt chain. When a PCI interrupt occurs, the user application is expected to execute one or more hooked ISP routines, including one of the above routines. The driver's ISP routine checks for DMA controller interrupts and local interrupts. DMA interrupts are handled internally by the driver. Local interrupts are handled by calling the *pci->local_isp( )* routine which must be provided by either a higher-level driver or by the user application.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Main.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080, tekpciPlx9080SetUserIo( )

PLX9080 Driver: DMA Functions

The **tekpciPlx9080** DMA functions, contained in **tekpciPlx9080Dma.c**, implement routines to perform DMA transfers using the PCI 9080 internal DMA controllers and to monitor the status of DMA transfers.

The current release of the drivers only supports DMA transfers in immediate mode (i.e. not using the linked-list queuing feature). Examples of linked-list queuing routines are included in **tekpciPlx9080Dma.c** but are not supported code in this revision of the driver.

To initiate a DMA transfer, the user application calls the **tekpciPlx9080DmaStart( )** routine with the appropriate parameters for the desired transfer addresses, length and direction. The driver will program the PCI 9080 DMA controller and start the DMA transfer. The user application may check on the status of the DMA transfer using the **tekpciPlx9080DmaIsDone( )** routine.

Typically, higher-level drivers implement application-specific DMA routines which call the lower-level PCI 9080 DMA routines to transfer data to and from the module. For example, if an FPMC-PIO module is being used, the **tekpciPioRxDmaStart** routine calls **tekpciPlx9080DmaStart** to perform the requested DMA transfer. If a higher-level driver is being used, the user application should call the higher-level DMA routines instead of the **tekpciPlx9080** DMA routines.

tekpciPlx9080DmaDisplay( )

NAME

tekpciPlx9080DmaDisplay( ) – Display DMA control registers.

SYNOPSIS

```
STATUS tekpciPlx9080DmaDisplay (
    TEKPCI_HANDLE pci,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine displays the DMA control registers for the PCI 9080 internal DMA controllers for the card specified by *pci* and displays a formatted status display using the user-supplied *output* and *handle* arguments. The DMA ISP counters are also displayed and cleared.

The DMA ISP counters are maintained only if the driver's ISP routine is used to handle PCI interrupts.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciPlx9080** driver.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Dma.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080

tekpciPlx9080DmaIsDone( )

NAME

tekpciPlx9080DmaIsDone( ) – Check to see if a DMA transfer has been completed.

SYNOPSIS

```
STATUS tekpciPlx9080DmaIsDone (
    TEKPCI_HANDLE pci,
    int channel
    );
```

DESCRIPTION

This routine checks to see if a DMA transfer has been completed. If the DMA controller "done" bit is set and the associated interrupt has been received, OK is returned. If the "done" bit is cleared or if the *pci->dma_done[channel]* field is zero, ERROR is returned.

Note that the *dma_done* field will only be set properly if the driver ISP is hooked into the PCI interrupt chain.

The *channel* argument specifies the DMA controller to check; for the PCI 9080, the *channel* argument must be between 0 and 1.

This routine may only be used for "direct" DMA control (i.e. without linked-list DMA control). Linked-list DMA control will be handled by a future version of the **tekpciPlx9080** driver.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Dma.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080, tekpciPlx9080DmaStart( )

tekpciPlx9080DmaStart( )

NAME

tekpciPlx9080DmaStart( ) – Start a DMA transfer.

SYNOPSIS

```
STATUS tekpciPlx9080DmaStart (
    TEKPCI_HANDLE pci,
    int channel,
    int read,
    int demand,
    void *pciaddr,
    WORD32 localaddr,
    int localincr,
    int length
    );
```

DESCRIPTION

This routine starts a DMA transfer using the PCI 9080 internal DMA controller. The DMA transfer is initiated in direct (i.e. non-linked-list) mode. The user application is responsible for ensuring that the DMA controller is idle prior to calling this routine.

The *channel* number specifies the DMA controller to use; for the PCI 9080, the *channel* argument must be between 0 and 1.

If *read* is non-zero, the DMA transfer reads data from the module to the PCI bus; otherwise, the DMA transfer writes data from the PCI bus to the module.

If *demand* is non-zero, the DMA controller's demand signal is enabled, allowing the module hardware to throttle DMA transfers. If *demand* is zero, the DMA transfer is performed without hardware pacing.

The *pciaddr* argument specifies the PCI bus address for the transfer. The *localaddr* argument specifies the module local bus address for the transfer. If *localincr* is non-zero, the local bus address is incremented during the transfer; otherwise, the local bus address is held constant.

The *length* argument specifies the length of the transfer in bytes.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Dma.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080, tekpciPlx9080DmaIsDone( )

PLX9080 Driver: EEPROM Functions

The **tekpciPlx9080** driver implements a set of functions to read and write the Serial EEPROM which is attached to the PCI 9080 controller.

The Serial EEPROM is implemented using a Fairchild DS93CS46 or DS93CS56 device. The DS93CS46 contains 64 16-bit words of data and the DS93CS56 contains 128 16-bit words of data.

The first N words of the Serial EEPROM are used for PCI 9080 register initialization. The value of N is determined by PCI 9080 configuration pins which are typically strapped to a required state on the hardware module. The table below shows the three possible PCI 9080 configurations.

Mode	# of 16-bit words	Description
Short Load	10	PCI 9080 SHORT# pin tied low.
Long Load	34	PCI 9080 SHORT# pin tied high and Local Bus Region Descriptor Register bit 25 (LBRDO[25]) is 0 in the Serial EEPROM.
Extra Long Load	42	PCI 9080 SHORT# pin tied high and LBRDO[25] is 1.

The remaining words in the Serial EEPROM are not used by the PCI 9080 and are available for other purposes. Typically, TEK's hardware products use the upper portion of the Serial EEPROM to record manufacturing information about the module. The upper portion is usually protected from inadvertent modification using the Serial EEPROM Protect Register feature. The number of words used for manufacturing information is module-dependent.

The middle portion of the Serial EEPROM is available for the user application if desired. Because the lower portion of the Serial EEPROM is not protected, a user application that uses the Serial EEPROM should take care to avoid modifying the values in the lower portion.

tekpciPlx9080EepromIsProtectEnabled( )

NAME

tekpciPlx9080EepromIsProtectEnabled( ) – Determine the state of the EEPROM Protect-Enable (PRE) jumper.

SYNOPSIS

```
STATUS tekpciPlx9080EepromIsProtectEnabled (
    TEKPCI_HANDLE pci,
    int *enabled,
    int *cleared
    );
```

DESCRIPTION

This routine attempts to determine whether the Serial EEPROM Protect Enable (“PRE”) jumper is installed and whether the Serial EEPROM is protected.

On some of TEK’s PMC/PCI modules, the FPGA can directly determine the state of the PRE jumper. On other modules, the driver performs EEPROM read operations to attempt to infer the state of the PRE jumper based on the data returned.

If the PRE jumper is installed, **enabled* is set to a non-zero value, otherwise, **enabled* is set to zero. If the PRE jumper is installed, **cleared* is updated with a non-zero value if the EEPROM protect register is cleared or set to zero if the EEPROM protect register has been programmed.

Either *enabled* or *cleared* may be NULL, in which case no update occurs to that variable.

When the PRE jumper is installed, the user may read or write the EEPROM Protect Register but cannot access the EEPROM data array. When the PRE jumper is not installed, the user may read or write the unprotected section of the EEPROM, but cannot access or modify the EEPROM Protect Register.

Refer to the Fairchild DS93CS46 and DS93CS56 data sheets for additional information about the function of the EEPROM Protect Register.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Eeprom.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080

tekpciPlx9080EepromRead( )

NAME

tekpciPlx9080EepromRead( ) – Read one word of Serial EEPROM data.

SYNOPSIS

```
STATUS tekpciPlx9080EepromRead (
    TEKPCI_HANDLE pci,
    int address,
    WORD16 *data
    );
```

DESCRIPTION

This routine reads one word (**tekpciPlx9080EepromRead**) from the Serial EEPROM attached to the PCI 9080 controller.

The routine fails if the Serial EEPROM Protect Enable (PRE) jumper is installed. To read the EEPROM Protect Register, use the routine **tekpciPlx9080EepromProtectRead( )**.

The first N Serial EEPROM words are used by the PCI 9080 controller for initialization; the value of N is module-dependent. See the PCI 9080 Data Book for a description of the Serial EEPROM format. These words should typically only be modified through the module-specific **Info** API (i.e. **tekpciPioInfo**).

The remaining EEPROM words are commonly used for module-specific parameters such as memory size and manufacturing options. These words should be accessed and interpreted through the appropriate module-specific **Info** API.

Address specifies the word number. For the DS93CS46 EEPROM, *address* may be between 0 and 63.

The data word read from the Serial EEPROM is copied to the buffer at **data*.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Eeprom.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080, tekpciPlx9080EepromReadN( )

tekpciPlx9080EepromReadN( )

NAME

tekpciPlx9080EepromReadN( ) – Read N words of Serial EEPROM data.

SYNOPSIS

```
STATUS tekpciPlx9080EepromReadN (
    TEKPCI_HANDLE pci,
    int address,
    int count,
    WORD16 *data
    );
```

DESCRIPTION

This routine reads a consecutive block of words from the Serial EEPROM attached to the PCI 9080 controller. This routine simply calls **tekpciPlx9080EepromRead( )** for each word to be read.

The routine fails if the Serial EEPROM Protect Enable (PRE) jumper is installed. To read the EEPROM Protect Register, use the routine **tekpciPlx9080EepromProtectRead( )**.

Address specifies the starting word number. For the DS93CS46 EEPROM, *address* may be between 0 and 63. *Count* specifies the number of words to read. *Count* must be greater than zero and the total of *address* and *count* must be less than the size of the EEPROM in words.

The data words read from the Serial EEPROM are copied to the buffer at **data*.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Eeprom.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080, tekpciPlx9080EepromRead( )

tekpciPlx9080EepromWrite( )

NAME

tekpciPlx9080EepromWrite( ) – Write one word of Serial EEPROM data.

SYNOPSIS

```
STATUS tekpciPlx9080EepromWrite (
    TEKPCI_HANDLE pci,
    int address,
    WORD16 data
    );
```

DESCRIPTION

This routine writes one word to the Serial EEPROM attached to the PCI 9080 controller.

The routine fails if the Serial EEPROM Protect Enable (PRE) jumper is installed. To write the EEPROM Protect Register, use the routine **tekpciPlx9080EepromProtectWrite( )**.

The first N Serial EEPROM words are used by the PCI 9080 controller for initialization; the value of N is module-dependent. See the PCI 9080 Data Book for a description of the Serial EEPROM format. The remaining EEPROM words are commonly used for module-specific parameters such as memory size and manufacturing options. These words should only be modified through the appropriate module-specific **Info** API (i.e. **tekpciPioInfo**)..

The Serial EEPROM locations which are used by the PCI 9080 or defined by the higher-level driver should typically only be modified through the module-specific **Info** API (i.e. **tekpciPioInfo**).

Address specifies the word number. For the DS93CS46 EEPROM, *address* may be between 0 and 63. *Data* specifies the word value to write.

The data value is not read back and verified; the user application should verify that the data word was written correctly. Attempts to write into protected regions will fail silently.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Eeprom.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080, tekpciPlx9080EepromWriteN( )

tekpciPlx9080EepromWriteN( )

NAME

tekpciPlx9080EepromWriteN( ) – Write N words of Serial EEPROM data.

SYNOPSIS

```
STATUS tekpciPlx9080EepromWriteN (
    TEKPCI_HANDLE pci,
    int address,
    int count,
    const WORD16 *data
    );
```

DESCRIPTION

This routine writes a consecutive block of words to the Serial EEPROM attached to the PCI 9080 controller. This routine simply calls **tekpciPlx9080EepromWrite( )** for each word to be written.

The routine fails if the Serial EEPROM Protect Enable (PRE) jumper is installed. To write the EEPROM Protect Register, use the routine **tekpciPlx9080EepromProtectWrite( )**.

Address specifies the first word number to write. For the DS93CS46 EEPROM, *address* may be between 0 and 63. *Count* specifies the number of words to write. *Count* must be greater than zero and the total of *address* and *count* must be less than the size of the EEPROM in words. The data words to write are located at **data*.

The data words written are not read back and verified; the user application should verify that the data words were written correctly. Attempts to write into protected regions will fail silently.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Eeprom.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080, tekpciPlx9080EepromWrite( )

tekpciPlx9080EepromWriteAll( )

NAME

tekpciPlx9080EepromWriteAll( ) – Send a Write All command to the Serial EEPROM, erasing all locations.

SYNOPSIS

```
STATUS tekpciPlx9080EepromWriteAll (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine sends a Write All opcode to the Serial EEPROM. The routine fails if the Serial EEPROM Protect Enable (PRE) jumper is installed.

This routine is equivalent to writing all locations with the value of 0xFFFF.

The write operation is not verified.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Eeprom.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080

tekpciPlx9080EepromWriteEnable( )

NAME

tekpciPlx9080EepromWriteEnable( ) – Send a Write Enable command to the Serial EEPROM.

SYNOPSIS

```
STATUS tekpciPlx9080EepromWriteEnable (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine sends a Write Enable opcode to the Serial EEPROM. The routine fails if the Serial EEPROM Protect Enable (PRE) jumper is installed.

This routine is used as a part of the sequence for modifying the EEPROM Protect Register. The Serial EEPROM requires a Write Enable with the PRE jumper removed before writing the Protect Register with **tekpciPlx9080EepromProtectWrite( )**. See the Fairchild DS93CS46 data sheet for more information on the Serial EEPROM protect features.

This routine is globally scoped to support manufacturing test and initialization; it should not be used by user application code.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Eeprom.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080

tekpciPlx9080EepromProtectClear( )

NAME

tekpciPlx9080EepromProtectClear( ) – Clear the Serial EEPROM Protect Register.

SYNOPSIS

```
STATUS tekpciPlx9080EepromProtectClear (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine clears the Serial EEPROM Protect Register. This routine fails if the Serial EEPROM Protect Enable (PRE) jumper is not installed; the Serial EEPROM does not allow the user to read or write the Protect Register without installing the PRE jumper.

If the PRE jumper is installed, a Protect Clear operation is performed. This operation both clears the Protect Register and also disables the EEPROM protect function, allowing all locations of the Serial EEPROM to be written.

Note that this is different than simply setting the Protect Register to zero, which would protect the entire Serial EEPROM array.

The Fairchild DS93CS46 Serial EEPROM uses the protect register to write-protect the words starting at the address programmed into the protect register. This feature is typically used in TEK Microsystems' products to protect the module-specific configuration information (contained in the last N words of the Serial EEPROM) from inadvertent modification.

This routine is globally scoped to support manufacturing test and initialization; it should not be used by user application code.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Eeprom.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080

tekpciPlx9080EepromProtectRead( )

NAME

tekpciPlx9080EepromProtectRead( ) – Read the Serial EEPROM Protect Register.

SYNOPSIS

```
STATUS tekpciPlx9080EepromProtectRead (
    TEKPCI_HANDLE pci,
    WORD16 *prdata
    );
```

DESCRIPTION

This routine reads the Serial EEPROM Protect Register. This routine fails if the Serial EEPROM Protect Enable (PRE) jumper is not installed; the Serial EEPROM does not allow the user to read or write the Protect Register without installing the PRE jumper. If the PRE jumper is installed, the contents of the Serial EEPROM Protect Register are copied to **prdata*.

The Fairchild DS93CS46 Serial EEPROM uses the protect register to write-protect the words starting at the address programmed into the protect register. This feature is typically used in TEK Microsystems' products to protect the module-specific configuration information (contained in the last N words of the Serial EEPROM) from inadvertent modification.

This routine is globally scoped to support manufacturing test and initialization; it should not be used by user application code.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Eeprom.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080

tekpciPlx9080EepromProtectWrite( )

NAME

tekpciPlx9080EepromProtectWrite( ) – Write the Serial EEPROM Protect Register.

SYNOPSIS

```
STATUS tekpciPlx9080EepromProtectWrite (
    TEKPCI_HANDLE pci,
    WORD16 prdata
    );
```

DESCRIPTION

This routine writes the Serial EEPROM Protect Register. This routine fails if the Serial EEPROM Protect Enable (PRE) jumper is not installed; the Serial EEPROM does not allow the user to read or write the Protect Register without installing the PRE jumper. If the PRE jumper is installed, the contents of the Serial EEPROM Protect Register are written with *prdata*.

The Fairchild DS93CS46 Serial EEPROM uses the protect register to write-protect the words starting at the address programmed into the protect register. This feature is typically used in TEK Microsystems' products to protect the module-specific configuration information (contained in the last N words of the Serial EEPROM) from inadvertent modification.

This routine is globally scoped to support manufacturing test and initialization; it should not be used by user application code.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Eeprom.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciPlx9080

PLX9080 Driver: Test Functions

The **tekpciPlx9080** driver implements Built-In-Test functions which support module confidence testing. The Built-In-Test functions may be used by user applications to verify basic operation of the hardware module.

Typically, a confidence test of a module consists of the following steps:

- As a part of initialization, verify the configuration space for the module. Modules which do not have valid configuration information will typically not be mapped to the **tekpciPlx9080** driver.
- Once the module has been attached to a driver, verify data bus functionality using the **tekpciPlx9080TestDatabus( )** function.
- If the **tekpciPlx9080** tests are successful, proceed with higher-level driver tests.

The **tekpciPlx9080** Built-In-Test interface also provides the utility function **tekpciPlx9080TestRegister( )** to test a single register location. This function is typically used by higher-level drivers to implement data bus tests of local module-specific registers.

tekpciPlx9080TestDatabus( )

NAME

tekpciPlx9080TestDatabus( ) – Test the PCI data bus to the PCI 9080 controller.

SYNOPSIS

```
STATUS tekpciPlx9080TestDatabus (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine performs a read-write test of a register in the PCI 9080 control/status register space using the **tekpciPlx9080TestRegister( )** function. This test verifies the PCI data path between the host processor and the PCI 9080 controller.

If *level* is **VERBOSE_NONE**, no display output is generated and the *output* and *handle* arguments may be NULL. If level is **VERBOSE_ERROR**, display output is generated only in the event of an error. If level is **VERBOSE_ALL**, display output is generated to report the results of the test.

Messages are displayed by calling *(*output) ( )* using the supplied *handle* argument.

This test uses PCI 9080 DMA address registers to perform the testing. This routine should not be invoked while either DMA controller is active.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciPlx9080** driver.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Test.c

RETURNS

OK, or ERROR if an error is encountered.

SEE ALSO

tekpciPlx9080, tekpciPlx9080TestRegister( )

tekpciPlx9080TestRegister( )

NAME

tekpciPlx9080TestRegister( ) – Test a single register location.

SYNOPSIS

```
STATUS tekpciPlx9080TestRegister (
    volatile WORD32 *ioptr,
    VERBOSE level,
    int usebytes,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine performs a read-write test of the 32-bit location at **ioreg*. The read-write test sequences through bit patterns which detect any stuck or shorted bits. The register location is set to zero at the end of the test.

If *usebytes* is non-zero, byte addressability of the register is also tested. If *usebytes* is zero, all accesses to the register are 32-bit word accesses.

If *level* is **VERBOSE_NONE**, no display output is generated and the *output* and *handle* arguments may be NULL. If level is **VERBOSE_ERROR**, display output is generated only in the event of an error. If level is **VERBOSE_ALL**, display output is generated to report the results of the test.

Messages are displayed by calling *(*output) ( )* using the supplied *handle* argument.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciPlx9080** driver.

INCLUDE FILES

tekpci.h, **tekpciPlx9080.h**

SOURCE FILES

tekpciPlx9080Test.c

RETURNS

OK, or ERROR if an error is encountered.

SEE ALSO

tekpciPlx9080, tekpciPlx9080TestDatabus( )

Flex Driver API

The **tekpciFlex** software driver serves as an abstraction layer between **tekpciPlx9080** and a higher level, in this case **tekpciLink**. The FLEX driver uses the **tekpciPlx9080** driver to talk to a PCI 9080 controller, read and write the EEPROM, read and write the I/O pins, and map the address spaces, procedures that are common to all current TEK PCI and PMC cards.

Given a **TEKPCI_HANDLE**, the FLEX driver handles downloading FPGAs using the **tekpciFlexFPGA** functions and manages information about the EEPROM using the **tekpciFlexInfo** functions along with the **TEKPCI_FLEX_INFO** data structure. The FLEX driver also recognizes and handles differences between VFLEX and standalone PCI/PMC modules. It does not, however, deal with application-specific features of the card or the application portion of the register space of the FPGAs.

The FLEX driver knows about the standard parts of the register space and components on the module and thus, using **tekpciFlexTest**, it is able to test the data buses, LED indicators, clock synthesizers and memory. One exception is that the driver does not know the number of clock synthesizers and relies on a higher level driver for that information.

These functions are typically invoked by a higher level driver and not by a user application.

The **tekpciFlex** driver is organized into the following files:

TekpciFlex.o	Library file which contains all of the following routines.
TekpciFlex.h	Header file with function prototypes for all external functions. This file should be #include'd by all user software that uses the **tekpciFlex** driver.
TekpciFlexIo.h	Header file with PCI 9080 I/O definitions. This file is used by the driver routines and may be #include'd by user routines which need to perform direct I/O to Flex registers.
TekpciFlexMain.c	Main driver file. Contains the following API functions: • tekpciFlexAttach • tekpciFlexDisplay • tekpciFlexFpgaIsLoaded • tekpciFlexFpgaGetDwg • tekpciFlexFpgaKill • tekpciFlexFpgaLoad • tekpciFlexFpgaLoadByDwg • tekpciFlexInit • tekpciFlexReset • tekpciFlexSetClockFreq

	• tekpciFlexSetLocalBusClockFreq
TekpciFlexInfo.c	Routines to manage reading and writing the PCI9080 serial EEPROM with appropriate data for the FLEX module. The actual EEPROM access is through the tekpciPlx 9080 driver. Contains the following API functions: • tekpciFlexInfoDisplay • tekpciFlexInfoGet • tekpciFlexInfoInit • tekpciFlexInfoSet • tekpciFlexInfoVerify
TekpciFlexTest.c	Routines to support BIT manufacturing test of the FLEX modules. Contains the following API functions: • tekpciFlexTestDatabus • tekpciFlexTestClock • tekpciFlexTestLed • tekpciFlexTestMemory

The following sections define the various global functions in the **tekpciFlex** driver. Functions are organized by file and major function.

Flex Driver: Basic Functions

The **tekpciFlex** basic functions contained in **tekpciFlexMain.c** implement routines to attach the FLEX driver to a FastPMC/FastPCI module and implement control and status functions which affect the FLEX driver as a whole.

The basic functions API consists of the following:

- **tekpciFlexAttach** to attach the FLEX driver to the **TEKPCI_HANDLE** data structure.
- **tekpciFlexDisplay** to display the FPGA control and status registers.
- **tekpciFlexInit** to initialize the FLEX driver.
- **tekpciFlexSetClockFreq** to set the clock synthesizer frequency of the bus on any channel supported by the hardware module.
- **tekpciFlexSetLocalBusClockFreq** to set the clock synthesizer frequency of the local bus clock.

tekpciFlexAttach( )

NAME

tekpciFlexAttach( ) – Attach the TEKPCI handle to the Flex driver.

SYNOPSIS

```
STATUS tekpciFlexAttach (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine attaches the TEKPCI handle to the FLEX driver by first using **tekpciPlx9080Attach( )** to attach the EEPROM. It then calls **tekpciFlexInfoGet( )** in order to obtain the information to set up the module specific fields of *pci*.

Unless there is a problem accessing the hardware, the routine will return OK. Otherwise it will return ERROR.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

ERROR if there is an error accessing hardware. OK otherwise.

SEE ALSO

tekpciFlex

tekpciFlexDisplay( )

NAME

tekpciFlexDisplay( ) – Displays the FPGA control and status registers.

SYNOPSIS

```
STATUS tekpciFlexDisplay (
    TEKPCI_HANDLE pci,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine reads the control and status registers of the FPGA for the hardware module specified by *pci*. The routine uses the user-supplied *output* and *handle* arguments to display a formatted status display.

This routine will only format a display after an FPGA has been loaded. If the FPGA is loaded the routine will return OK, otherwise it will return ERROR.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciFlex** driver.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

OK if an FPGA is loaded, ERROR otherwise.

SEE ALSO

tekpciFlex

tekpciFlexInit( )

NAME

tekpciFlexInit( ) – Initializes the driver's internal data structures

SYNOPSIS

```
STATUS tekpciFlexInit (
    void
    );
```

DESCRIPTION

This routine initializes the internal data structures of the driver by using **tekpciPlx9080( )** to initialize the PLX 9080 driver and then using **tekFpgaModelInstall( )** to install a default FPGA.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

ERROR if driver can not be initialized. OK otherwise.

SEE ALSO

tekpciFlex, tekpciPlx9080( ), tekFpgaModelInstall( )

tekpciFlexSetClockFreq( )

NAME

tekpciFlexSetClockFreq( ) – Sets clock synthesizer.

SYNOPSIS

```
STATUS tekpciFlexSetClockFreq (
    TEKPCI_HANDLE pci,
    int channel,
    int *freq
    );
```

DESCRIPTION

This routine programs the ICD2053 clock synthesizer in the hardware module indicated by *pci*. The routine assumes that the FPGA is loaded and compliant.

The hardware module is assumed to have N clock synthesizers, with 0 … N-2 mapped to channels 0 to N-2 and clock synthesizer N-1 mapped to the local bus. If *channel* is –1, it maps to the local bus, otherwise, *channel* must be between 0 and N-1 where N is the number of non-local-bus synthesizers supported by the module.

The frequency with which the clock synthesizer is to be programmed is specified by the variable **freq*. **freq* is also modified by the routine to reflect the actual frequency finally programmed into the clock synthesizer.

If an invalid *channel* or **freq* is supplied, the routine will return ERROR. Additionally, if there is a problem accessing the hardware, the routine will also return ERROR. Otherwise it returns OK.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

ERROR if given an invalid parameter or there is a problem accessing hardware. OK otherwise.

SEE ALSO

tekpciFlex

tekpciFlexSetLocalBusClockFreq( )

NAME

tekpciFlexSetLocalBusClockFreq( ) – Sets the clock synthesizer frequency of the local bus clock.

SYNOPSIS

```
STATUS tekpciFlexSetLocalBusClockFreq (
    TEKPCI_HANDLE pci,
    int *freq
    );
```

DESCRIPTION

This routine sets the ICD2053 local bus clock synthesizer on the hardware module specified by *pci*. It assumes that the FPGA is loaded and compliant.

If the hardware module does have a variable clock synthesizer for the local bus, it uses **tekpciFlexSetClockFreq( )** to set the local bus clock to the frequency specified by **freq*. Otherwise, **freq* must be the same as the local bus frequency or the function will return ERROR.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

ERROR if given invalid parameters or there is a problem accessing hardware. OK otherwise.

SEE ALSO

tekpciFlex, tekpciFlexSetClockFreq( )

Flex Driver: FPGA Functions

The **tekpciLink** FPGA functions, contained in **tekpciFlexMain.c** and **tekpciFlexFpga.c**, implement routines to download FPGA images to the module, identify the FPGA image loaded into the module, and access standard FPGA images which are built into the **tekpciFlex** driver.

The **tekpciFlex** driver implements a library of FPGA images with associated ID values. The ID value for an FPGA image combines the drawing number, FPGA size, and revision information to uniquely identify a specific FPGA image.

The full FPGA ID value consists of a five digit drawing number, two digit FPGA size, and two digit revision number. For example, the standard streaming I/O FPGA for a HOTLINK PMC module is drawing number 23623, FPGA size 50, and revision 01, making the ID value 236235001.

When looking up FPGA images, the revision field affects the search as follows: if the revision value is non-zero, the specific revision number is used; otherwise, the highest available revision is used. For example, if the driver has FPGA images 236235001 (rev -) and 236235002 (rev A) defined, the user application can use the rev - image by using an ID value of 236235001 or can use the latest available revision by using an ID value of 236235000.

If the user application has its own FPGA image to download, the user application can call the **tekpciFlexFpgaLoad( )** routine with a pointer to the FPGA image (a **const TEK_FPGA_ENTRY** pointer). Typically, this requires that the user application include an FPGA image as a C language include file and build an appropriate **TEK_FPGA_ENTRY** data structure for the FPGA image.

The FPGA functions are as follows:

- **tekpciFlexFpgaGetDwg,** which gets the drawing and revision of the loaded FPGA.
- **tekpciFlexFpgaIsLoaded**, which checks if there is an FPGA loaded.
- **tekpciFlexFpgaKill**, which resets the FPGA image in preparation for a new one.
- **tekpciFlexFpgaLoad**, which loads a given FPGA image
- **tekpciFlexFpgaLoadByDwg**, which looks up and loads an FPGA
- **tekpciFlexReset**, which resets the FPGA's internal logic.

Using these functions, if the user application needs to download a driver-supplied FPGA image, the user can look up an image and then load it using **tekpciFlexFpgaLoad( )**, or call the **tekpciLinkFpgaLoadByDwg( )** routine as a shortcut.

tekpciFlexFpgaGetDwg( )

NAME

tekpciFlexFpgaGetDwg( ) – Get the FPGA drawing and revision information.

SYNOPSIS

```
STATUS tekpciFlexFpgaGetDwg (
    TEKPCI_HANDLE pci,
    int *dwg,
    int *rev
    );
```

DESCRIPTION

This routine checks to see if an FPGA image is loaded into the FastPMC/FastPCI module specified by *pci* (using **tekpciFlexFpgaIsLoaded( )**) and then reads the FPGA drawing and revision information from the ID register in the local address space.

If the FPGA is not loaded or if a hardware error is encountered, ERROR is returned. If the FPGA is loaded and the ID register is valid, OK is returned.

The drawing number portion of the ID register is copied into **dwg* and the revision number portion of the ID register is copied into **revision*. Either *dwg* or *revision* may be NULL, in which case no update is performed to that variable.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

OK if an FPGA is loaded, ERROR otherwise.

SEE ALSO

tekpciFlex, tekpciFlexIsFpgaLoaded( )

tekpciFlexFpgaIsLoaded( )

NAME

tekpciFlexFpgaIsLoaded( ) – Check whether FPGA is loaded.

SYNOPSIS

```
STATUS tekpciFlexFpgaIsLoaded (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine checks the FPGA device on the FastPMC/FastPCI module specified by *pci* and returns OK if the FPGA is loaded with a program image and ERROR otherwise.

The FPGA program image is detected by examining the FLEX 10K CONF_DONE signal. This will indicate whether a program image has been successfully downloaded; no checking is performed on whether the image is accessible through the local bus.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

OK if an FPGA is loaded, ERROR otherwise.

SEE ALSO

tekpciFlex, tekpciFlexFpgaLoad( )

tekpciFlexFpgaKill( )

NAME

tekpciFlexFpgaKill( ) – Reset FPGA image.

SYNOPSIS

```
STATUS tekpciFlexFpgaKill (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine resets the FPGA image on the FastPMC/FastPCI module specified by *pci*. This causes the FPGA to exit user mode and await a new configuration image; all local bus operations will be invalid until a new image is downloaded using **tekpciFlexFpgaLoad( )**.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciFlex

tekpciFlexFpgaLoad( )

NAME

tekpciFlexFpgaLoad( ) – Load an FPGA image.

SYNOPSIS

```
STATUS tekpciFlexFpgaLoad (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    const TEK_FPGA_ENTRY *fep,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine loads the FPGA image specified by *fep* into the FastPMC/FastPCI module specified by *pci* and returns OK if the FPGA is loaded successfully and ERROR otherwise.

The *level* argument determines the amount of status display generated by the function. If *level* is **VERBOSE_NONE**, no status output is generated and output and handle may be NULL. If *level* is **VERBOSE_ERROR**, status output will be generated only in the case of an error. If *level* is **VERBOSE_ALL**, status output will be generated as the FPGA is downloaded.

All output messages are generated by calling *output* with *handle* as an argument.

INCLUDE FILES

tekpci.h, **tekpciFlex.h, tekFpga.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

OK if an FPGA is loaded, ERROR otherwise.

SEE ALSO

tekpciFlex, tekpciFlexFpgaLoadByDwg( ), tekpciFlexFpgaIsLoaded( )

tekpciFlexFpgaLoadByDwg( )

NAME

tekpciFlexFpgaLoadByDwg( ) – Lookup and then load an FPGA image by ID value.

SYNOPSIS

```
STATUS tekpciFlexFpgaLoadByDwg (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    int dwg,
    int revision,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine looks up the FPGA image specified by *dwg* and *revision* and then, if the FPGA image is found, calls **tekpciFlexFpgaLoad( )** to load the FPGA image into the FastPMC/FastPCI module specified by *pci*. If the lookup and load operations are both successful, OK is returned; otherwise, ERROR is returned.

The *dwg* argument is the drawing number of the desired FPGA image. The *dwg* value may be determined by the user or may be generated from an FPGA name by calling **tekpciFlexFpgaLookupId( )**. The dwg and revision arguments are combined with the FPGA size on the FastPMC/FastPCI module to generate the FPGA image ID.

The *level* argument determines the amount of status display generated by the function. If *level* is **VERBOSE_NONE**, no status output is generated and output and handle may be NULL. If *level* is **VERBOSE_ERROR**, status output will be generated only in the case of an error. If *level* is **VERBOSE_ALL**, status output will be generated as the FPGA is downloaded.

All output messages are generated by calling *output* with *handle* as an argument.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

OK if an FPGA is loaded, ERROR otherwise.

SEE ALSO

tekpciFlex, tekpciFlexFpgaLoad( ), tekpciFlexFpgaIsLoaded( )

tekpciFlexReset( )

NAME

tekpciFlexReset( ) – Reset FPGA internal logic.

SYNOPSIS

```
STATUS tekpciFlexReset (
    TEKPCI_HANDLE pci
    );
```

DESCRIPTION

This routine resets the FPGA on the FastPMC/FastPCI module specified by *pci*. The FPGA reset is performed by writing the internal reset control bit. The semantics of the reset function are FPGA-dependent; the typical FPGA resets the RX and TX physical FIFOs along with all internal state machines and control registers.

The routine then enables the TX outputs, waits for a minimum of 1 millisecond, and then clears all pending error bits.

The user must have previously loaded an FPGA image to the card using the **tekpciFlexFpgaLoad( )** or **tekpciFlexFpgaLoadByDwg( )** routines.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexMain.c

RETURNS

OK, or ERROR if an error is encountered accessing the hardware.

SEE ALSO

tekpciFlex

Flex Driver: Information Functions

The **tekpciFlex** information functions contained in **tekpciFlexInfo.c** implement routines to download, verify, and update information stored on the EEPROM with the use of the lower level **tekpciPLX9080** driver.

The **tekpciFlexInfo** functions use the **TEKPCI_FLEX_INFO** data structure to relay information about the specified PCI/PMC device between the FLEX driver and the calling function. It contains the necessary information to fully identify a device.

The **TEKPCI_FLEX_INFO** structure is shown bellow:

```
typedef struct tekpciFlexInfo {
   int model;
   int serialnum;              /* Typical: 1998390001  */
   char modelname[80];
   int model_backend;          /* Model # of backend (0 if monolithic) */
   int serialnum_backend;      /* Serial # of back end (0 if monolithic) */
   char modelname_backend[80];
   int hw_revision;            /* Typical: 1 */
   int memory_size             /* 0 or 1024 (in KB) */
   int fpga_family;            /* 0 = 10k, 1 = 10KA, 2 = 10KV, 3 = 10 KE */
   int fpga_size;              /* 30 = 10K30, 50 = 10K50 */
   int fpga_speed;             /*1 to 3 = dash-1 to dash-3 */
   int module_info[16];        /* Module-specific information */
} TEKPCI_FLEX_INFO;
```

The information functions API consists of the following:

- **tekpciFlexInfoDisplay** to display the information about a particular PCI/PMC device.
- **tekpciFlexInfoGet** to get the information about a FLEX device.
- **tekpciFlexInfoInit** to initialize the FLEX driver.
- **tekpciFlexInfoSet** to update the serial EEPROM information.
- **tekpciFlexInfoVerify** to confirm that the information on a EEPROM is correct.

tekpciFlexInfoDisplay( )

NAME

tekpciFlexInfoDisplay( ) – Displays information about PCI/PMC device.

SYNOPSIS

```
STATUS tekpciLinkFpgaLookupId (
    TEKPCI_FLEX_INFO infop,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine displays information about a PCI/PMC device by using the user-supplied *output* and *handle* arguments to display a formatted version of the data given by *infop*.

This routine does not access the hardware for information, but instead assumes that **tekpciFlexInfoGet( )** has already been called.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciFlex** driver.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexInfo.c

RETURNS

OK

SEE ALSO

tekpciFlex, tekpciFlexInfoGet( )

tekpciFlexInfoGet( )

NAME

tekpciFlexInfoGet( ) – Gets information about a FLEX device.

SYNOPSIS

```
STATUS tekpciFlexInfoGet (
    TEKPCI_HANDLE pci,
    TEKPCI_FLEX_INFO *infop
    );
```

DESCRIPTION

This routine reads the EEPROM data using the PCI 9080 routine **tekpciPlx9080EepromReadN( )** and extracts the PCI/PMC specific information. The PCI/PMC information is then used to fill in the data structure pointed to by *infop*.

The information used for the data structure pointed to by *infop* comes from words 55 through 63 of the data read using **tekpciPlx9080EepromReadN( )**. Both older formats and current formats of these words are compatible with this function.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexInfo.c

RETURNS

ERROR if there is a problem accessing hardware. Otherwise, OK

SEE ALSO

tekpciFlex, tekpciPlx9080EepromReadN( )

tekpciFlexInfoInit( )

NAME

tekpciFlexInfoInit( ) – Initializes the serial EEPROM.

SYNOPSIS

```
STATUS tekpciFlexInfoInit (
    TEKPCI_HANDLE pci,
    int model,
    int submodel,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine generates a set of PCI 9080 configuration words and writes the words to the serial EEPROM using the PCI 9080 driver. The first 44 words are written; i.e. this is a PCI 9080 “Extra Long EEPROM Load”.

This routine will return ERROR if there is any problem accessing the hardware. It also verifies the words written to the EEPROM and will return an ERROR if they are incorrect.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

TekpciFlexInfo.c

RETURNS

ERROR if there are any problems accessing and writing to hardware. Otherwise OK.

SEE ALSO

tekpciFlex, tekpciPlx9080

tekpciFlexInfoSet( )

NAME

tekpciFlexInfoSet( ) – Sets EEPROM data.

SYNOPSIS

```
STATUS tekpciFlexInfoSet (
    TEKPCI_HANDLE pci,
    TEKPCI_FLEX_INFO *infop
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine updates the EEPROM data with data based on the information contained in the data structure pointed to by *infop*. All access to the EEPROM is made via the PLX 9080 driver.

This routine modifies only words 55 through 63 inclusive of the serial EEPROM. It does not modify words 0 through 54.

This routine will return ERROR if there is a problem accessing hardware or if any of the *infop* fields are not valid. ERROR will also be returned if the routine is unable to verify the data written to the EEPROM. If there is a problem accessing the EEPROM, the function uses the *output* and *handle* variables to display more specific information about where the error occurred.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexInfo.c

RETURNS

ERROR if there is a problem accessing the hardware or if given invalid parameters. OK otherwise.

SEE ALSO

tekpciFlex, tekpciPlx9080

tekpciFlexInfoVerify( )

NAME

tekpciFlexInfoVerify( ) – Verifies data from the serial EEPROM

SYNOPSIS

```
STATUS tekpciFlexInfoVerify (
    TEKPCI_HANDLE pci,
    int model,
    int submodel,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine verifies a set of PCI 9080 configuration words by reading data from the serial EEPROM and comparing it to the expected EEPROM configuration words. All access to the EEPROM is done using the PLX 9080 driver.

The expected EEPROM configuration words are static and not user supplied with the exception of the *model* and *submodel*.

All of the PCI 9080 words are verified, including the first 44. This makes it a PCI 9080 “Extra Long EEPROM Load.”

This routine will return ERROR if there is a problem accessing the EEPROM. If this occurs, the function uses the user-supplied *output* and *input* to display a more specific error message.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexInfo.c

RETURNS

ERROR if there is a problem accessing hardware or if verify failed. Otherwise OK.

SEE ALSO

tekpciFlex, tekpciPlx9080

Flex Driver: Test Functions

The **tekpciFlex** functions contained in **tekpciFlexTest.c** implement built-in BIT and manufacturing tests of the FLEX module to support module confidence testing. The built-in test functions may be used by the user application in order to verify the basic operation of the hardware module.

The test functions API consists of the following:

- **tekpciFlexTestClock** to test the clock synthesizer and local-bus clock.
- **tekpciFlexTestDatabus** to test the data buses to the PCI 9080 controller.
- **tekpciFlexTestLed** to test the LED indicators on the module.
- **tekpciFlexTestMemory** to test the RX and TX FIFO buffer memory.

tekpciFlexTestClock( )

NAME

tekpciFlexTestClock( ) – Tests the clock.

SYNOPSIS

```
STATUS tekpciFlexInfoVerify (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    int channel,
    void (*output) (void *handle, const char *s)
    void *handle
    );
```

DESCRIPTION

This routine tests the clock synthesizer and local-bus clock for the channel indicated by *channel* on the hardware module indicated by *pci*.

In order to verify the clocks' correctness, the routine tests both clocks by programming the clock at a specific frequency, waiting for it to settle, and then reading the FPGA clock counter and comparing that with the clock counter. This test is executed at multiple frequencies.

The local-bus clock is restored to its correct running frequency upon completion of the routine.

If *level* is **VERBOSE_NONE**, no display output is generated and the *output* and *handle* arguments may be NULL. If *level* is **VERBOSE_ERROR**, display output is generated only in the event of an error. If *level* is **VERBOSE_ALL**, display output is generated to report the results of the test. All display outputs are achieved using the user-supplied *output* and *handle.*

The routine returns ERROR if there is a problem accessing the hardware or the clock fails the test and the clock counter does not match the FPGA clock. Otherwise, the routine returns OK.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciFlex** driver.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexTest.c

RETURNS

Ok, or ERROR if there is a problem accessing hardware or the clock fails the test.

SEE ALSO

tekpciFlex

tekpciFlexTestDatabus( )

NAME

tekpciFlexTestDatabus( ) – Test the PCI data bus to PCI 9080 controller and the local data bus to the PCI 9080 and the FPGA.

SYNOPSIS

```
STATUS tekpciFlexTestDatabus (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    void (*output) (void *handle, const char *s)
    void *handle
    );
```

DESCRIPTION

This routine first performs a test of the data bus between the host and the PCI 9080 using **tekpciPlx9080TestDatabus( )** and then tests the local data bus between the PCI 9080 and the FPGA using a read-write test.

If *level* is **VERBOSE_NONE**, no display output is generated and the *output* and *handle* arguments may be NULL. If *level* is **VERBOSE_ERROR**, display output is generated only in the event of an error. If *level* is **VERBOSE_ALL**, display output is generated to report the results of the test. All display outputs are achieved using the user-supplied *output* and *handle.*

This test uses the PCI 9080 DMA address registers and thus should not be invoked while either DMA controller is active.

If the test is successfully passed, the routine will return OK. Otherwise it returns ERROR.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciFlex** driver.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexTest.c

RETURNS

OK if test passed. ERROR otherwise.

SEE ALSO

tekpciFlex, tekpciPlx9080

tekpciFlexTestLed( )

NAME

tekpciFlexTestLed( ) – Tests LEDs present on device

SYNOPSIS

```
STATUS tekpciFlexTestLed (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    void (*delay) (void),
    void (*output) (void *handle, const char *s),

    void *handle
    );
```

DESCRIPTION

This routine performs a test of any and all LED indicators present on the FLEX device indicated by the *pci* variable.

For each indicator, the routine will change its state from Off to each of its possible output colors in turn (typically green and then red) with a delay between each state. The specific LED sequence is module dependant. The delay is implemented by calling the *delay* function which is assumed to provide a one second delay.

This routine distinguishes between only two levels of verbose. If *level* is **VERBOSE_ALL** then it will specify if there are no LED indicators present. All levels will cycle through all LED indicators and display the expected behavior. The display output is generated using the user-supplied *output* and *handle* arguments.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciFlex** driver.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexTest.c

RETURNS

OK

SEE ALSO

tekpciFlex

tekpciFlexTestMemory( )

NAME

tekpciFlexTestMemory( ) – Performs test of RX and TX FIFO buffers.

SYNOPSIS

```
STATUS tekpciFlexTestMemory (
    TEKPCI_HANDLE pci,
    VERBOSE level,
    void (*output) (void *handle, const char *s),
    void *handle
    );
```

DESCRIPTION

This routine performs a memory test on the RX and TX FIFOs. This is done by writing to the TX FIFO and verifying what is received by the RX FIFO.

If *level* is **VERBOSE_NONE**, no display output is generated and the *output* and *handle* arguments may be NULL. If *level* is **VERBOSE_ERROR**, display output is generated only in the event of an error. If *level* is **VERBOSE_ALL**, display output is generated to report the results of the test. All display outputs are achieved using the user-supplied *output* and *handle.*

If there is a problem accessing hardware or the FIFOs fail the test, this routine will return ERROR. Otherwise it will return OK.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekpciFlex** driver.

INCLUDE FILES

tekpci.h, **tekpciFlex.h**

SOURCE FILES

tekpciFlexTest.c

RETURNS

ERROR if test failed or problem accessing hardware. OK otherwise.

SEE ALSO

tekpciFlex

ClockSyn Driver API

The **tekClockSyn** driver consists of functions which support the clock synthesizers used on the hardware module. Currently, this is only an ICD2053 clock synthesizer.

The **tekClockSyn2053** functions contained in **tekClockSyn.c** compute the program words for an ICD2053 clock synthesizer and program the appropriate control/program words to the device. The routines are also intended to be hardware-independent.

The functions in the ClockSyn driver use the data structure **TEK_CLKSYN** for storing and passing data about a clock. It is structured as follows:

```
typedef struct tekClksyn {
   WORD32 ref_freq;
   WORD32 req_freq;
   WORD32 act_freq;
   WORD32 ctl_word;
   volatile WORD32 *ctlreg;
   WORD32 ctl_sclk;
   WORD32 ctl_sdata;
   int spincount_bit;
   int spincount_settling;
} TEK_CLKSYN;
```

These functions are typically invoked by a higher level driver and not by a user application.

The **tekClockSyn** driver is organized into the following files:

tekClockSyn.o	Library file which contains all of the following routines.
tekClockSyn.h	Header file with function prototypes for all external functions. This file should be #include'd by all user software that uses the **tekClockSyn** driver.
tekClockSyn.c	Main driver file. Contains the following API functions: • tekClockSyn2053Compute • tekClockSyn2053Describe • tekClockSyn2053Set

ClockSyn Driver: 2053 Functions

The **tekClockSyn2053** functions provide the ability to compute, display, and write the program words needed to control an ICD2053 clock synthesizer. For correctness, **tekClockSyn2053Compute( )** must be called before any attempt to display or write clock settings.

- **tekClockSyn2053Compute** to find the actual clock frequency and control word.
- **tekClockSyn2053Describe** to display information about the clock synthesizer.
- **tekClockSyn2053Set** to write the control words to the clock synthesizer and set it.

tekClockSyn2053Compute( )

NAME

tekClockSyn2053Compute( ) – Computes program words for an ICD2053 clock synthesizer.

SYNOPSIS

```
STATUS tekClockSyn2053Compute (
    TEK_CLKSYN *clkp
    );
```

DESCRIPTION

This routine takes *clkp* and uses information in the data structure to which it points to compute the actual frequency at which the ICD2053 clock synthesizer will run and to construct the appropriate control word. The control word and actual frequency are placed in the data structure pointed to by *clkp*.

This routine will return OK unless the requested frequency is in an invalid range. If this occurs, it will return ERROR.

INCLUDE FILES **tekpci.h, tekClockSyn.h**

SOURCE FILES **tekClockSyn2053.c**

RETURNS ERROR if given invalid parameter. OK otherwise.

SEE ALSO **tekClockSyn**

tekClockSyn2053Describe( )

NAME

tekClockSyn2053Describe( ) – Displays information about the clock synthesizer.

SYNOPSIS

```
STATUS tekClockSyn2053Describe (
    TEK_CLKSYN *clkp,
    char *buf
    );
```

DESCRIPTION

This routine creates a formatted display of information contained in the control word stored in the data structure pointed to by *clkp*.

The routine uses **sprintf** to put the formatted display into *buf*.

The output formatting is intended to be used to provide the user with a status display. The output formatting is not considered an API interface and may change arbitrarily in future releases of the **tekClockSyn** driver.

INCLUDE FILES

tekpci.h, **tekClockSyn.h**

SOURCE FILES

tekClockSyn2053.c

RETURNS

none

SEE ALSO

tekClockSyn

tekClockSyn2053Set( )

NAME

tekClockSyn2053Set( ) – Sets ICD2053 clock synthesizer.

SYNOPSIS

```
STATUS tekClockSyn2053Set (
    TEK_CLKSYN *clkp
    );
```

DESCRIPTION

This routine uses the control words contained in the data structure pointed to by *clkp* to set the ICD2053 clock synthesizer.

In order to have valid control words, this routine should only be called after **tekClockSyn2053Compute( )**.

INCLUDE FILES

tekpci.h, **tekClockSyn.h**

SOURCE FILES

tekClockSyn2053.c

RETURNS

OK

SEE ALSO

tekClockSyn, tekClockSyn2053Compute

FPGA Driver API

The FPGA Driver consists of routines which support the installation of FPGA images into a master table and then the lookup of these images by ID. This driver is also used to keep track of the various standard FPGAs that are installed by the different **TEKPCI** drivers.

There are two sets of functions, **tekFpgaImage** and **tekFpgaModel**. **tekFpgaImage** functions use an image table to handle the FPGA images and references them primarily by ID number. **tekFpgaModel** functions use a model table and are primarily useful for obtaining information about the FPGAs corresponding to specific models.

These functions are typically invoked by a higher level driver and not by a user application.

The **tekFpga** driver is organized into the following files:

tekFpga.o	Library file which contains all of the following routines.
tekFpga.h	Header file with function prototypes for all external functions. This file should be #include'd by all user software that uses the **tekFpga** driver.
tekFpgaMain.c	Main driver file. Contains the following API functions: • tekFpgaImageGet • tekFpgaImageInstall • tekFpgaImageLookup • tekFpgaModelGet • tekFpgaModelInstall

The following sections define the various global functions in the **tekFpga** driver. Functions are organized by file and major function.

FPGA Driver: Image Functions

The **tekFpgaImage** functions are used to install and keep track of the various standard FPGA images. This is done using an FPGA image table, which maintains a list of the FPGAs and the necessary information for each one.

An FPGA in the table can be accessed using a specified FPGA ID number. The ID value for an FPGA image combines the drawing number, FPGA size and revision information to uniquely identify a specific FPGA image.

The full FPGA ID value consists of a five digit drawing number, two digit FPGA size, and two digit revision number. For example, the standard streaming I/O FPGA for a HOTLINK PMC module is drawing number 23623, FPGA size 50, and revision 01, making the ID value 236235001.

When looking up FPGA images, the revision field affects the search as follows: if the revision value is non-zero, the specific revision number is used; otherwise, the highest available revision is used. For example, if the driver has FPGA images 236235001 (rev -) and 236235002 (rev A) defined, the user application can use the rev - image by using an ID value of 236235001 or can use the latest available revision by using an ID value of 236235000.

The basic FPGA functions are:

- **tekFpgaImageGet** to get information about an FPGA image given an ID.
- **tekFpgaImageInstall** to install FPGA images into the FPGA image table.
- **tekFpgaImageLookup** to return information about an FPGA given its name.

tekFpgaImageGet( )

NAME

tekiFpgaImageGet( ) – Gets the FPGA image for the specified ID.

SYNOPSIS

```
STATUS tekFpgaImageGet (
    int id,
    int *model,
    int *revision,
    const char **name,
    const TEK_FPGA_ENTRY **fep
    );
```

DESCRIPTION

This routine looks up the appropriate FPGA image for the specified ID value in the driver library of FPGA images.

By convention, the ID value consists of a five digit drawing number, two digit FPGA size and two digit revision. For example, FPGA drawing 23620, in a 10K30, revision 02 would have an ID value of 236203002.

If *id* has a non-zero revision field, the specified revision is required. If *id* has a revision field of zero, the latest available revision will be returned.

If a matching FPGA image is available, a pointer to the FPGA data structure is copied into **fep* and OK is returned. If no matching FPGA image is available, ERROR is returned.

This routine simply accesses a table of FPGA images in the driver. It does not access or modify the FastPMC/FastPCI module or limit the lookup to appropriate FPGA images for the installed hardware.

INCLUDE FILES

tekpci.h, **tekFpga.h**

SOURCE FILES

tekFpgaMain.c

RETURNS

OK if an FPGA image is available, ERROR otherwise.

SEE ALSO

tekFpga

tekFpgaImageInstall( )

NAME

tekiFpgaImageInstall( ) – Installs an FPGA image into the image

table.

SYNOPSIS

```
STATUS tekFpgaImageGet (
    const char *name,
    int model,
    int revision,
    const TEK_FPGA_ENTRY * fep
    );
```

DESCRIPTION

This routine installs FPGA images into the **tekFpgaImageTable** in order that they can be used by other **tekFpga** routines. This is done by using the user-provided variables *name*, *model*, *revision*, and *fep* to create a new entry in **tekFpgaImageTable** if there is space available.

If **tekFpgaImageTable** is full, ERROR will be returned. Otherwise the routine will return OK.

INCLUDE FILES **tekpci.h**, **tekFpga.h**

SOURCE FILES **tekFpgaMain.c**

RETURNS ERROR if the image table is full. OK otherwise.

SEE ALSO **tekFpga**

tekFpgaImageLookup( )

NAME

tekiFpgaImageLookup( ) – Given the name, model, and revision, returns the FPGA entry in the FPGA table.

SYNOPSIS

```
STATUS tekFpgaImageLookup (
    const char *name,
    int *model,
    int *revision,
    int index,
    const char **outname,
    const TEK_FPGA_ENTRY ** fep
    );
```

DESCRIPTION

This routine looks up the specified FPGA by *name*. The name is used along with the model and revision to find a matching FPGA entry in the image table. *Model*, *revision*, *fep* and *outname* are then filled using the data from the table entry.

The variable *index* is the number of items in the table. ??I think??

This routine will return OK if the FPGA has been inserted into the image table. This should be done using **tekFpgaImageInstall( )**. If the FPGA can not be found, it returns ERROR.

INCLUDE FILES

tekpci.h, **tekFpga.h**

SOURCE FILES

tekFpgaMain.c

RETURNS

OK if an FPGA image is found, ERROR otherwise.

SEE ALSO

tekFpga, tekFpgaImageInstall( )

FPGA Driver: Model Functions

The **tekFpgaModel** functions are used to keep track of the various models and the standard FPGAs used with them.

The **tekFpgaModel** uses a model table, which consists of the model, part number, and the FPGA ID. This table can then be used to find the standard FPGA ID of a given model. Using the FPGA ID, the FPGA can then be accessed by functions that perform FPGA functions based on the ID. **tekFpgaModelInstall( )** should be used prior to accessing the FPGA model table.

The FPGA model functions are:

- **tekFpgaModelGet** to get part number and name for a given model.
- **tekFpgaModelInstall** to install a model into the model table.

tekFpgaModelGet( )

NAME

tekiFpgaModelGet( ) – Get the part number and name for a given model.

SYNOPSIS

```
STATUS tekFpgaImageGet (
    int model,
    const char **partnum,
    const char **name
    );
```

DESCRIPTION

This routine takes the user-supplied *model* and uses it to look up the corresponding part number and name which are stored in *partnum* and *name* respectively.

In order to successfully find a given model, the model must have first been installed using **tekFpgaModelInstall( )** since it uses the model table.

If the requested model is available, the routine returns OK. Otherwise it returns ERROR.

INCLUDE FILES

tekpci.h, **tekFpga.h**

SOURCE FILES

tekFpgaMain.c

RETURNS

OK if the FPGA model is available, ERROR otherwise.

SEE ALSO

tekFpga, tekFpgaModelInstall( )

tekFpgaModelInstall( )

NAME

tekFpgaModelInstall( ) – Replace the specified ID.

SYNOPSIS

```
STATUS tekpciFpgaImageGet (
    int model,
    const char *partnum,
    const char *name,
    );
```

DESCRIPTION

This routine takes the user-supplied *model*, *partnum*, and *name* and installs them into the model table, **tekFpgaModelTable**. This routine must be called before any other accesses to the model table in order to not get ERROR.

If there are too many entries in the model table and it is full, the routine will return ERROR. Otherwise it will return OK.

INCLUDE FILES

tekpci.h, **tekFpga.h**

SOURCE FILES

tekFpgaMain.c

RETURNS

ERROR if the model table is full, OK otherwise.

SEE ALSO

tekFpga

PCI Configuration API

The file **tekpciVxWorks.c** implements the generic PCI services which are required to configure and map the PCI/PMC card into the target processor's address space. These functions are necessarily BSP-specific, as there is no standard VxWorks API for accessing PCI configuration registers.

The code in **tekpciVxWorks.c** has been tested on a Motorola MVME2604 PowerPC carrier card and a VMETRO MIDAS-120R Intel i960 carrier card. The code should work unmodified on other Motorola MVME-series carrier cards and any other PowerPC-based carriers which follow Motorola's API convention for PCI configuration space access.

On the Motorola MVME2604 platform, the VxWorks BSP provides low-level routines to read and write the PCI configuration space. These routines are prototyped in the VxWorks file **`drv/pci/pciIomapLib.h`** and are listed below.

```
/* Read PCI configuration space */

STATUS pciConfigInByte (int busNo, int deviceNo, int funcNo,
                        int address, char * pData);
STATUS pciConfigInWord (int busNo, int deviceNo, int FuncNo,
                        int address, short * pData);
STATUS pciConfigInLong (int busNo, int deviceNo, int FuncNo,
                        int address, int * pData);

/* Write PCI configuration space */

STATUS pciConfigOutByte (int busNo, int deviceNo, int funcNo,
                         int address, char data);
STATUS pciConfigOutWord (int busNo, int deviceNo, int funcNo,
                         int address, short data);
STATUS pciConfigOutLong (int busNo, int deviceNo, int funcNo,
                         int address, int data);
```

The driver uses these routines to access PCI configuration space to determine the module type by inspecting the manufacturer and module ID fields.

The MIDAS version of `tekpciVxWorks.c` implements these six routines by calling the appropriate routines in the MIDAS BSP supplied by VMETRO.

```
/* TEK functions (used by TEKPCI drivers) */

STATUS tekpciConfigOutByte (TEKPCI_HANDLE pci, int index,
                            WORD8 value);
STATUS tekpciConfigOutWord (TEKPCI_HANDLE pci, int index,
```

```
                                WORD16 value);
    STATUS tekpciConfigOutLong (TEKPCI_HANDLE pci, int index,
                                WORD32 value);

    STATUS tekpciConfigDisplay (TEKPCI_HANDLE pci,
                                void (*output) (void *handle,
                                                const char *msg),
                                void *handle);

    STATUS tekpciHeaderGet (TEKPCI_HANDLE pci,
                            PCI_HEADER_DEVICE *hdr);

    STATUS tekpciHeaderDisplay (TEKPCI_HANDLE pci,
                                void (*output) (void *handle,
                                                const char *msg),
                                void *handle);

STATUS tekpciMap (TEKPCI_HANDLE pci, WORD32 RegIoAddrCard,
                                WORD32 RegMemAddrCard,
                                WORD32 MemAddrCard1,
                                WORD32 MemAddrCard2
                                );

    /* Block/restore interrupts */

    int tekpciIntDisable (TEKPCI_HANDLE pci);
    int tekpciIntRestore (TEKPCI_HANDLE pci, int level);
```

The following sections describe the tekpciVxWorks.c functions.

Demo Application (Command Line Driven)

TEK has implemented a demonstration application that supports a command line interface to the driver functions. The demonstration application is built using a preexisting set of monitor routines that are not fully documented here.

The demonstration application may be executed by opening the WindSh application connected to the VxWorks target and executing the command:

```
< start.<bspname>
```

This will load the **tekpciPlx9080.obj, tekpciFlex.obj** and **tekpciPio.obj** libraries, load the *demo.o* demonstration application, and run the demo application by executing *root( ).* The demo application will display a startup message followed by a command prompt on the target console.

The following table lists some of the commands that are available in the demo application. Many of these commands perform low-level operations on the hardware module for factory test and diagnostic purposes and are not intended for general purpose use.

Command	Description
HELP	Lists help on some commands. Not fully implemented.
USAGE <command>	Lists command token list for the specified command. May also be invoked using "?" as an argument (i.e. "DM ?").
EXIT	Terminate monitor, return to calling shell.
DM[.type] [address] [length]	Display memory. Type may be B (byte), W (word), L (longword), S (swapped longword).
PM[.type] <address> [value…]	Patch memory. Type may be B (byte), W (word), L (longword), S (swapped longword). If invoked with values on the command line, PM performs exactly one write per location with the values specified. If invoked in the interactive mode (no values on the command line), PM reads the address, presents the value, and accepts a response. In the interactive mode, the user may enter an optional value, following by either =, /, or . An '=' writes the value and rereads the current address. A '/' writes the value and steps to the previous address. A '.' writes the value and terminates the PM command. A <CR> writes the value and steps to the next address. If no value is entered, the write operation does not occur.
PCI [STATUS]	Display PCI address parameters for the default PCI handle (**tekpciPlx9080Display**).
PCI DMA	Display PCI DMA information.
PCI INT	Display PCI interrupt information.

Command	Description
PCI RELOAD	Reload PCI state from EEPROM; this deletes the current configuration space settings.
PCI REGTEST	Perform PCI driver register test (**tekpciPlx9080TestDatabus**).
PCI EEPROM [STATUS]	Display PCI Serial EEPROM status.
PCI EEPROM READ <offset>	Read PCI Serial EEPROM contents (**tekpciPlx9080EepromRead**).
PCI EEPROM WRITE <offset> <data>	Write PCI Serial EEPROM contents (**tekpciPlx9080EepromWrite**).
PCI EEPROM DUMP	Display PCI Serial EEPROM contents.
PCI EEPROM ERASE	Erase PCI Serial EEPROM contents.
PCI EEPROM WRALL	Erase PCI Serial EEPROM contents using Write All.
PCI EEPROM WREN	Send Write Enable command to PCI Serial EEPROM.
PCI EEPROM PRCLEAR	Clear PCI Serial EEPROM Protect Register.
PCI EEPROM PRREAD	Read PCI Serial EEPROM Protect Register
PCI EEPROM PRWRITE <data>	Write PCI Serial EEPROM Protect Register
PCI REG <offset>	Display PCI 9080 control/status register.
PCI SETREG <offset> <value>	Set PCI 9080 control/status register.
PCI SETCONF <offset> <value>	Set PCI configuration space register.
PIO [STATUS]	Display PIO driver status (**tekpciPioDisplay**).
PIO PARMS	Display PIO driver parameters (**tekpciPioParmsDisplay**).
PIO KILLFLEX	Restart hardware FPGA (**tekpciPioFpgaKill**).
PIO FPGA <dwg>	Load FPGA image (**tekpciPioFpgaLoad**)
PIO SETCLOCK <freq>	Set TX clock frequency (**tekpciPioSetTxParms**).
PIO READ	Read PIO RX FIFO and display results until FIFO is empty (**tekpciPioRxGetPtr**, **tekpciPioRxGetCount**).
PIO FLUSH	Read PIO RX FIFO and discard data until FIFO is empty (**tekpciPioRxGetPtr**, **tekpciPioRxGetCount**).
PIO WRITE <value…>	Write PIO TX FIFO with specified values (**tekpciPioTxGetFree**, **tekpciPioTxGetPtr**).
PIO IDLE	Set PIO to idle mode (disable RX and TX).
PIO RX	Set PIO to RX mode (enable RX).
PIO TX	Set PIO to TX mode (enable TX).
PIO TXCLK <keyword>	Set PIO TX clock source (SYNTH, BUSCLK or P2)
PIO REGTEST [passes]	Perform PIO driver register test (**tekpciPioTestDatabus**).

Command	Description
PIO CLKTEST [passes]	Perform PIO driver clock test (**tekpciPioTestClock**).
PIO MEMTEST [passes]	Perform PIO driver memory test (**tekpciPioTestMemory**). The memory test performed is dependent on the FPGA image that has been downloaded.
PIO LOOP [count] [passes]	Perform PIO driver loopback test (requires two PIO cards).
PIO DMATXLOOP [count] [passes]	Perform PIO driver loopback test using DMA for TX only (requires two PIO cards).
PIO DMARXLOOP [count] [passes]	Perform PIO driver loopback test using DMA for RX only (requires two PIO cards).
PIO INFO	Display PIO EEPROM information (tekpciPioInfoDisplay).